Федеральное государственное бюджетное учреждение науки
Институт динамики систем и теории управления
Сибирского отделения Российской академии наук

На правах рукописи

Фереферов Евгений Сергеевич

ТЕХНОЛОГИЯ АВТОМАТИЗАЦИИ СОЗДАНИЯ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ С ГИС-ФУНКЦИОНАЛЬНОСТЬЮ НА ОСНОВЕ ИХ ДЕКЛАРАТИВНЫХ СПЕЦИФИКАЦИЙ

05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата технических наук

Научный руководитель:
академик, д.т.н. Бычков И.В.

Иркутск – 2014

Содержание

Содержание 2
Введение 4
Глава 1. Технологии и инструментальные средства разработки прикладных программных систем 13
1.1. Императивное программирование и библиотеки компонентов 13
1.2. Порождающее программирование и CASE системы 15
1.3. Декларативное программирование 17
1.4. Методы интеграции ГИС-функциональности в разрабатываемые системы 19
1.5. Сравнение инструментальных средств для задачи разработки приложений баз данных 20
Выводы 30
Глава 2. Технология автоматизации создания приложений БД 31
2.1. Предлагаемый подход 31
2.2. Модель приложения БД 37
2.3. Язык спецификаций приложений БД 43
Выводы 48
Глава 3. Инструментальная система создания приложений БД. 49
3.1. Требования к функциональному обеспечению и архитектуре инструментальной системы 49
3.2. Ядро системы 52
3.3. Подсистема управления спецификациями 61
3.4. Подсистема Редактор БД 66
3.5. Построитель пользовательских запросов 70
3.6. Программный интерфейс для взаимодействия с внешними программными системами 74
3.7.Построитель отчётов 77
3.8. Подсистема «Карта» 79
Выводы 84

Глава 4.Применение инструментальной системы «ГеоАРМ» 85
4.1.Разработка приложения для БД «Pubs» 85
4.2. Разработка АИС «Управление многоквартирными домами г. Иркутска» ... 92
ЗАКЛЮЧЕНИЕ 116
Список сокращений и условных обозначений 118
Литература 119
Приложение А. Семантика языка спецификаций ПБД 129
Приложение Б. Спецификация АИС «Единый общегородской регистр адресов недвижимости» 141

Введение

Актуальность темы. Современные прикладные программные системы (ППС) ориентированы на решение вычислительных задач или задач обработки данных. В обоих случаях, как правило, осуществляется работа с большими объёмами информации сложной структуры, что обуславливает необходимость использования баз данных (БД) и реализации специализированных функций для решения задач конкретной предметной области. В их числе сбор, хранение, и преобразование информации, выполнение вычислений над полем данных, а также агрегация, анализ и предоставление данных пользователям. Создание эффективных процедур для работы с БД является важной задачей при проектировании и реализации ППС. Необходимость обработки и визуализации разнообразных форм информации (текстовой, числовой, графической) требует при реализации ППС разработки качественных человеко-машинных интерфейсов и привлечения в ряде случаев специализированных технологий (например, геоинформационных технологий для обработки данных, обладающих пространственной привязкой).

Программное обеспечение, реализующее интерфейс для взаимодействия пользователей с предметными БД, будем называть приложением баз данных (ПБД). Основные задачи ПБД: обеспечение выполнения стандартных операций, а именно, создания, чтения, модификации и удаления записей таблиц БД, а также ряда дополнительных, таких, как поиска, фильтрации данных, формирования отчетов, и других операций. ПБД могут быть как частью (подсистемой) ППС (например, частью биллинговой системы), так и самостоятельными ППС (например, городской реестр адресов, реестр объектов недвижимости). Кроме того, ПБД могут входить в состав вычислительных систем, например, в качестве подсистемы для регистрации пользователей.

Современные технологии разработки ПБД, основанные на императивном программировании и использовании развитых библиотек визуальных компонентов (например, Visual Component Library [45,90] или Microsoft

Foundation Classes [43] или Framework Class Library [5]) предоставляют общецелевые инструментальные средства. Среди них имеются компоненты, реализующие как части визуального интерфейса, так и бизнес-логики для доступа и модификации БД. При разработке ПБД, как правило, сначала создаётся структура (схема) БД, а затем ПБД (иногда эти процессы могут происходить одновременно). При этом доступ к каждой индивидуальной таблице реализуется при помощи одних и тех же (или сходных по функциям) подпрограмм. При изменении структуры БД, например, добавлении новых таблиц или полей, необходимо вносить изменения в программный код ПБД, заново реализуя функции для доступа к новым элементам БД. Невысокий уровень автоматизации реализации однотипных функций для работы с таблицами БД приводит к большими временны́м и трудовым затратам при создании (модернизации) ПБД.

Существующие подходы в области объектно-реляционного отображения [71], (например, Hibernate/NHibernate [77] или Entity Framework [79]) позволяют ускорить разработку ПБД за счёт автоматизации построения объектной модели обеспечивающей взаимодействие с сущностями реляционной БД и являющейся описанием структуры БД для приложения. При этом часть программного кода, в частности классы соответствующие сущностям БД, классы обеспечивающие преобразование объектных запросов в SQL-запросы, генерируется автоматически. Необходимо отметить, что изменения в структуре БД приводят к необходимости перегенерации классов, что не является сложным процессом, но требует аккуратного программирования при доработке этих классов, чтобы не смешивать автоматически сгенерированный и созданный вручную код. Дальнейшая реализация системы, т.е. программирование пользовательского интерфейса, операций поиска, и т.д. уже выполняется вручную отдельно для каждой сущности, только теперь через свойства и методы классов, что снова приводит к созданию больших объёмов однотипного кода.

В настоящее время активно ведутся исследования в области автоматизации разработки как пользовательских интерфейсов (например, Model-Based User Interface Development), так и ППС в целом (например, Model Driven

Architecture [22], порождающее программирование [64]), позволяющих повысить эффективность процесса создания приложений. Основной тенденцией в этих исследованиях является разработка современных методов структурирования метаданных (данных о структуре) об ППС в виде моделей системы (иногда только пользовательского интерфейса [23,74,87]) различными средствами (например, надстройки над моделями классов UML [65], применение модельных каркасов [66], или построения онтологий предметной области [24,25,38]). Формализация знаний о структуре ППС в модели позволяет выделять схожие структуры данных и присущие им бизнес-процессы в отдельные компоненты, а также генерировать соответствующие им сценарии создания структур в СУБД, алгоритмы обработки бизнес-процессов, экранные формы единожды и распространить их на все подобные компоненты. Как показывает практика, сгенерированный код практически всегда требуется дорабатывать программисту, при этом полученные изменения не отражаются в исходных абстрактных моделях системы.

Для решения задач обработки, представления и анализа пространственных данных (ПД) современные ППС должны включать соответствующие функциональные возможности геоинформационных систем (ГИС). Современные ГИС предоставляют разработчикам API (Application Programming Interface) для реализации ГИС-функциональности [21]. Несмотря на развитость этих API, реализация таких функций – сложная и трудоёмкая задача, требующая знаний в области геоинформационных технологий. Реализация ГИС-функциональности существующими методами часто приводит к дублированию функций целевой ГИС в разрабатываемой системе. Модернизация существующих ППС, направленная на интеграцию с функциями обработки ПД, как правило, требует наличия исходных кодов этих ППС у разработчика, а в случае их отсутствия приводит к необходимости повторной разработки системы.

Анализ существующих подходов к созданию ПБД показывает невысокий уровень автоматизации, а реализация существующих технологий не может исключить этап доработки программного кода непосредственно разработчиками.

Это обосновывает актуальность задачи разработки концептуально новых технологий и инструментальных средств, автоматизирующих процесс создания ПБД, которые позволят сократить сроки и, как следствие, затраты на создание (модернизацию) ПБД, а также позволят решать более широкий круг комплексных научно-исследовательских задач за счёт встроенной ГИС-функциональности. В качестве способа представления и хранения модели приложений БД в работе предложено использовать декларативные спецификации [1] – детальные описания в текстовом виде структур приложений, требований к функциональности, правил представления и обработки данных и механизмов взаимодействия с внешними ППС. Декларативные спецификации отличаются своей компактностью по сравнению с программами на императивных языках, обладают предметной ориентированностью и выразительностью, а также возможностью интерпретации различными трансформационными и другими процедурами [27].

Цель и задачи исследования. Целью диссертационной работы является разработка языка и инструментального средства формирования декларативных спецификаций ПБД, а также технологии применения этих спецификаций для автоматизации процесса создания и модернизации ПБД, обладающих ГИС-функциональностью.

Основные задачи исследования.

1. Провести анализ существующих подходов к автоматизации создания ППС.
2. Разработать технологию автоматизации создания ПБД, обладающих ГИС-функциональностью, на основе их декларативных спецификаций.
3. Построить концептуальную модель ПБД, описывающую структуру БД, правила отображения схемы БД в пользовательский интерфейс и механизм взаимодействия ПБД с внешними ППС, в том числе с ГИС.
4. Создать декларативный язык спецификации ПБД.
5. Реализовать инструментальное средство для автоматизации создания ПБД, обладающих ГИС-функциональностью, на основе их декларативных спецификаций.

6. В качестве апробации результатов диссертационного исследования применить разработанную технологию для решения ряда практических задач.

Объектом исследования являются технологии автоматизации создания ПБД.

Предметом исследования являются декларативные спецификации и их использование для автоматизации создания ПБД с ГИС-функциональностью.

Методы исследования. Для решения поставленных задач использованы методы объектно-ориентированного, модульного и сборочного программирования, построения интерпретаторов, баз данных, разработки предметно-ориентированных языков, спецификаций программ и автоматизации создания ППС.

Научная новизна работы заключается в следующем.

Предложена технология автоматизации создания ПБД, отличием которой от известных является выделение информации о структуре ПБД и формирование спецификаций в виде формализованных знаний. Это позволяет: абстрагироваться от структуры БД и типа используемой СУБД, использовать универсальные алгоритмы для доступа и модификации таблиц БД, динамического создания пользовательского интерфейса, а также взаимодействия с внешними ППС, в том числе с ГИС.

Создана оригинальная концептуальная модель ПБД, особенность которой заключается в том, что информация о структуре БД расширена знаниями о способах представления данных пользователю и механизме взаимодействия с внешними ППС, что позволяет создавать универсальные алгоритмы для взаимодействия с таблицами БД, динамического формирования пользовательского интерфейса и взаимодействия с внешними системами, в том числе с ГИС.

Разработан новый предметно-ориентированный декларативный язык спецификаций ПБД, включающий конструкции для описания не только структур таблиц и связей между ними, но и правил формирования пользовательского

интерфейса для взаимодействия с этими таблицами, взаимосвязи информации из БД с пространственными данными, а также механизма взаимодействия с внешними ППС, решающими специфические задачи.

Впервые разработано инструментальное средство, позволяющее интерактивно создавать спецификации ПБД, а также настраиваться при помощи спецификаций на работу с предметной БД.

В целом, в диссертации предложена новая технология создания ПБД, обладающих ГИС-функциональностью, позволяющая значительно сократить сроки и стоимость разработки, а также снизить требования к квалификации разработчика в части знания языков программирования за счёт применения интерактивного инструментального средства для создания спецификаций ПБД.

Научная и практическая значимость результатов. Основные научные результаты по теме диссертации получены в рамках следующих программ и проектов: программы фундаментальных исследований СО РАН (проект IV.31.2.4. «Методы и технологии разработки программного обеспечения для анализа, обработки и хранения разноформатных междисциплинарных данных и знаний, основанные на применении декларативных спецификаций форматов представления информации и моделей программных систем», 2010-2012 гг.); программы фундаментальных исследований Отделения нанотехнологий и информационных технологий РАН (проект № 3, 2009-2011 гг., № 4.1, 2012-2014 гг.); междисциплинарной программы 4.5.2. СО РАН (проект 4.5.2.1. «Интеллектные методы и инструментальные средства создания и анализа интегрированных распределённых информационно-аналитических и вычислительных систем для междисциплинарных исследований с применением ГИС, GRID– и Веб– технологий» 2007-2009 гг.); междисциплинарного интеграционного проекта СО РАН (проект № 121) и проектов РФФИ (08-07-00163-а, 09-07-12017-офи_м, 11-07-00426-а, 11-07-92204-Монг_а).

Разработанные в рамках диссертационной работы технология и инструментальное средство позволяют значительно повысить эффективность, снизить трудозатраты и сократить сроки создания ПБД, обладающих ГИС-

функциональностью. Созданное программное обеспечение зарегистрировано в Федеральной службе по интеллектуальной собственности, патентам и товарным знакам [12,14,16]. Практическая значимость результатов определяется их использованием при создании интегрированных прикладных программных систем: «Муниципальная ГИС г. Иркутска», «Муниципальная информационная система градостроительной деятельности г. Иркутска», АИС «Управление многоквартирными домами», АИС «Отдел жилищного хозяйства», АИС «Топонимика г. Иркутска», АИС «Реестр геодезических съёмок», АИС «Единый общегородской регистр адресов объектов недвижимости».

Соответствие специальности. В соответствии с формулой специальности 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей», диссертационная работа охватывает решение задач создания и сопровождения программных средств различного назначения и включает исследование моделей, методов и алгоритмов проектирования программ и программных систем, языков и программных инструментов для организации взаимодействия программ и программных систем, систем управления базами данных, человеко-машинных интерфейсов.

Научное и народнохозяйственное значение диссертации заключается в повышении эффективности процессов обработки данных в вычислительных машинах, комплексах и компьютерных сетях, а также в сокращении сроков их создания.

Отражённые в диссертации положения соответствуют пунктам 3 и 7 области исследования специальности 05.13.11.

Апробация работы. Основные результаты диссертационной работы, её отдельные положения, а также результаты конкретных прикладных исследований и разработок обсуждались на научных семинарах ИДСТУ СО РАН, на международных, региональных научных и научно-практических конференциях и совещаниях, на научных координационных советах СО РАН и на региональных совещаниях, посвященных методам и технологиям информатики, развитию

информационно-телекоммуникационной инфраструктуры и внедрению геоинформационных систем в управление территориальным развитием.

Результаты диссертационной работы докладывались и обсуждались на следующих научных конференциях: VI, VIII, IX Байкальских школах-семинарах «Математическое моделирование и информационные технологи» (г. Иркутск, 2005, 2006, 2007 гг.); X, XII, XIV, XVII, XVIII Байкальских Всероссийских конференциях «Информационные и математические технологии в науке и управлении» (г. Иркутск 2005, 2007, 2009, 2012, 2013 гг.); II, III Международных конференциях «Инфокоммуникационные и вычислительные технологии и системы» (г. Улан-Удэ – о. Байкал, 2006, 2010 гг.); «Ляпуновские чтения» (г. Иркутск, 2006–2013 гг.); Школа-семинар молодых ученых «Информационные технологии и моделирование социальных эколого-экономических систем» (Россия, г. Иркутск – Монголия, п. Ханх, 2008, 2013 гг.); 3-я Всероссийская конференция «Винеровские чтения» (г. Иркутск, 2009 г.); Сибирский научно-практический семинар «Информационные технологии регионального и муниципального управления» (г. Барнаул, 2009 г.); Всероссийская конференция «Математическое моделирование и вычислительно-информационные технологии в междисциплинарных научных исследованиях» (г. Иркутск, 2011 г.); Международная конференция «Математические и информационные технологии. МИТ 2011» (Сербия, г. Врнячка Баня, Черногория, г. Будва, 2011 г.); 17-я международная научная конференция «Системный анализ, управление и навигация» (г. Евпатория, 2012 г.).

Публикации. По теме диссертации опубликованы 11 статей в рецензируемых журналах из перечня ВАК, 3 в специальных выпусках периодических журналов, 1 монография, а также получены 3 свидетельства об официальной регистрации программ для ЭВМ Федеральной службы по интеллектуальной собственности, патентам и товарным знакам.

Личный вклад автора. Все выносимые на защиту научные положения получены соискателем лично. В основных научных работах по теме диссертации, опубликованных в соавторстве, лично соискателем разработаны: в

[10,11, 15, 47, 53-55] – технология создания ПБД, обладающих ГИС-функциональностью на основе спецификаций, [49, 53, 55, 57, 58] – концептуальная модель ПБД, [15, 57, 60] – язык спецификаций ПБД, [8, 11, 12, 15, 20, 51, 55, 56] – инструментальное средство создания ПБД на основе декларативных спецификаций, [46, 48-50] – программная реализация картографического модуля, [14, 48]– библиотека для создания отчетов, [15] – прикладная АИС «Управление многоквартирными домами г. Иркутска», реализованная в рамках предложенной технологии.

Разработка алгоритмов обработки данных и программная реализация модуля для работы с БД получены в неделимом соавторстве с к.т.н. Хмельновым А.Е. Программная реализация всех упомянутых в диссертации библиотек, модулей и систем выполнена в рамках обозначенных ранее научно-исследовательских работ в отделе «Комплексных информационных систем» ИДСТУ СО РАН при непосредственном участии автора.

Структура и объем диссертации. Диссертационная работа состоит из введения, четырех глав, заключения и списка литературы, включающего 92 наименований, и приложений. Объем составляет 152 страницы, включая 117 страниц основного текста, 38 рисунок, 10 таблиц, список сокращений и условных обозначений, словарь терминов и понятий.

Глава 1. Технологии и инструментальные средства разработки прикладных программных систем

В настоящее время существует большое количество технологий и инструментальных средств, позволяющих в различной степени автоматизировать создание ППС для хранения и обработки тематических и пространственных данных. По парадигме разработки технологии разделяют на модульные, сборочные и порождающие, по способу программирования – на императивные, декларативные, функциональные. Существуют технологии разработки, которые интегрируют в себе несколько разных подходов и инструментальных средств (например, реализация технологии MDA – Bold for Delphi [22, 85]).

Выбор той или иной инструментальной технологии создания ППС зависит от используемой разработчиками методологии [39]. Необходимо отметить, что современные методологии ориентированы на итерационно-инкрементальную разработку, что позволяет поэтапно развивать интерфейс системы и наращивать ее функциональные возможности. При этом важным требованием к технологии является возможность быстрого создания рабочего прототипа ППС для его демонстрации заказчикам, уточнения требований и последующего развития.

Рассмотрим некоторые технологии и инструментальные средства с точки зрения средств автоматизации разработки более подробно.

1.1. Императивное программирование и библиотеки компонентов

Одним из старейших подходов к созданию программ является императивное программирование. Данный подход предполагает последовательное выполнение инструкций для манипулирования данными. Императивное программирование соответствует алгоритмическому решению задач, в которых последовательное исполнение команд является естественным. В качестве средств структурирования в императивном программировании

применяется метод выделения частей программного кода, решающих определенную подзадачу, в виде подпрограмм (процедур, функций). Кроме того, выделение реализации отдельных алгоритмов в подпрограммы дает возможность обращаться к ним многократно и тем самым ускорить процесс разработки.

Развитием средств автоматизации разработки в императивном программировании являются подходы «модульное программирование» и «структурное программирование». Суть модульного программирования состоит в разбиении всей программы на группы компонентов, называемых модулями, каждый из которых направлен на решение определенной обособленной задачи и имеет интерфейс для взаимодействия с внешней средой (другими программными блоками). При этом каждый модуль отвечает требованиям: блочной организации, синтаксической обособленности, семантической независимости, общности данных и полноты определения (аксиома модульности Коуэна). Обособленность и независимость программных модулей позволяет безболезненно заменять их подобными (при условии идентичности интерфейсов модулей) и в целом сказывается на скорости разработки ППС. Создание программ на основе готовых модулей называют «сборочным программированием» [63].

Методы структурного программирования позволяют облегчить и ускорить разработку ПО за счет соблюдения ряда правил организации создания программного кода. Основные принципы структурного программирования – это нисходящая пошаговая декомпозиция общей задачи к более мелким, разбиение программы на модули и структурное кодирование – применение только структур типа «Следования», «Если-то-иначе», «Цикл с предусловием» (теорема о структурировании Бойма и Джокопини [75]). Большой вклад в теорию и практику создания систем на основе программных модулей и пактов прикладных программ сделали такие ученые, как Ершов А.П. [28], Опарин Г.А. [36], Тыугу [44].

При создании программных систем важную роль играют не только вопросы организации программного кода, но и средства поддержки разработки, позволяющие облегчить и автоматизировать процессы конструирования, компиляции и отладки. Такими инструментальными средствами являются

интегрированные среды разработки ПО (IDE – Integrated development environment).

Современные IDE частично используют принципы концепции RAD (Rapid Application Development) [83] и позволяют автоматизировать процесс создания приложений за счет консолидации средств конструирования, отладки программ и библиотек готовых компонентов (например, VCL, MFC) в одной инструментальной системе. Обычно IDE включает в себя текстовый редактор [18], компилятор и/или интерпретатор [3, 61], средства автоматизации сборки и отладчик [81]. IDE часто содержит разнообразные инструменты для упрощения конструирования графического интерфейса пользователя, а так же невизуальные компоненты для сборки из готовых компонент алгоритмической части. Для поддержки объектно-ориентированной разработки ПО [6, 68, 92] современные среды разработки включают браузер классов, инспектор объектов и диаграмму иерархии классов. Хотя и существуют среды разработки, предназначенные для нескольких языков, такие как Eclipse или Microsoft Visual Studio [32], обычно среда разработки предназначается для одного определённого языка программирования, как например, Visual Basic [40], Embarcadero Delphi [29,37], Embarcadero C++, NetBeans [34]. Большинство IDE могут применяться вне зависимости от выбранной методологии разработки. Но некоторые методологии все же ориентированы на применение конкретных IDE (например, CDM и Oracle Developer Suite).

1.2. Порождающее программирование и CASE системы

В последнее время активно развивается подход к автоматическому созданию информационных систем, называемый «порождающее программирование». В основе данной парадигмы лежит идея трансформации моделей высокого уровня, описывающих предметную область создаваемых информационных систем, в низкоуровневые модели реализации (исходный код на конкретном языке программирования). В качестве средства формирования и

представления модели предметной области часто используют Unified Modeling Language (UML) [7]. Данный язык позволяет создавать графические спецификации ППС, описывающие как структуру, так и поведение системы. Хорошо известен такой способ моделирования предметной области как создание онтологии [31]. Например, в работах [23, 65] авторы строят онтологии предметной области и генерируют по ним пользовательский интерфейс ППС, в работах [33,38] на основе онтологий предметных областей создаются экспертные системы в области безопасности в энергетике и производстве. Необходимо отметить работы [9,30], в которых разработан широкий спектр практически значимых методов и средств синтеза программ в рамках исследований по созданию пакетов прикладных программ.

Высокая скорость создания программ в порождающем программировании достигается за счет сборки программного кода из отдельных компонентов по заданным требованиям к синтезируемой системе. Причем, в отличие от методологии компонентного программирования, в порождающем программировании происходит автоматическая сборка программного продукта.

Синтез ПО в данном подходе базируется на трех составляющих: наборе (библиотеке) компонент реализации, модели знаний о трансляции абстрактных требований в конкретные связи компонентов и генераторе, реализующем эти знания. Как правило, компоненты реализации ориентированы на определенную архитектуру разрабатываемого ПО (например, ПБД). Модель знаний о трансляции включает задаваемый разработчиком набор правил преобразования объектов абстрактной модели ППС в компоненты реализации. Генератор на основе абстрактной модели и знаний трансляции осуществляет автоматическую сборку (синтез) целевого кода из готовых компонент реализации. В большинстве случаев при разработке достаточно сложных систем в подходе «порождающее программирование» создается только так называемый каркас ППС, поэтому сгенерированный код обычно требует доработки вручную.

Важным элементом технологии порождающего программирования является инструмент поддержки процесса проектирования и разработки абстрактных

моделей разрабатываемых ППС. В качестве таких инструментальных средств часто применяются различные CASE-системы (Computer-Aided Software Engineering) [17, 35].

CASE-системы обеспечивают поддержку следующих процессов создания и сопровождения ППС: анализ и формулировку требований, проектирование БД и модулей ППС, генерацию кода, тестирование, документирование, контроль качества, управление конфигурацией и проектом в целом. Важным преимуществом использования CASE-систем при разработке ППС является наличие графических средств моделирования предметной области, что позволяет разработчикам наглядно изучать разрабатываемую систему, изменять её согласно поставленным целям и задачам. Большинство существующих CASE-средств (например, Sybase PowerDesigner или IBM Rational Software) основано на методологиях структурного или объектно-ориентированного анализа и проектирования, в которых для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств используются спецификации в виде диаграмм (как правило на UML) или текстов.

1.3. Декларативное программирование

Декларативное программирование – парадигма программирования, в которой, в отличии от императивного программирования, для решения конкретной задачи описывается её структура и что должно быть получено в результате без описания способа решения. Например, в коде web-страницы на HTML [91] описывается содержание страницы (текст, элементы интерфейса) без описания способа её отображения. Язык запросов SQL [26] описывает, что должно быть выбрано из таблиц БД, а как это будет реализовано, решает конкретная СУБД.

Важной особенностью декларативных языков является отсутствие в них команд реализации алгоритмов (создания, модификации конкретных объектов),

что дает возможность абстрагироваться от конкретной реализации и создавать платформо-независимые системы. К преимуществам парадигмы декларативного программирования относят [80] простоту написания программ, легкость восприятия кода программистами (по сравнению с императивными программами), простота сопровождения и модернизации. Часто декларативные языки применяются для создания спецификаций [1]. Спецификации – это средство описания, формализации задач, программ или ППС в целом.

Выразительными средствами спецификаций могут быть не только декларативные языки. Известны различные графические средства создания спецификаций, в том числе UML (например, в работе [69] предложен диаграммный язык для создания спецификаций пользовательских интерфейсов). Спецификации, для создания которых применяют декларативные языки, будем называть «декларативными спецификациями».

Важными свойствами спецификаций являются *точность, понятность и полнота* [1, 2, 19]. Свойство полноты требует присутствия в спецификации всей значимой для решения конкретной задачи информации, что гарантирует получение адекватного результата. Точность спецификации обеспечивает однозначную интерпретацию описанных в ней объектов. В силу понятности декларативные спецификации просты для восприятия человеком, кроме того, спецификации легко тестируются, поскольку все связи между объектами программы статичны, хорошо поддаются трансформации и на их основе могут быть сгенерированы приложения. Поэтому спецификации удобно использовать для описания структур пользовательских интерфейсов, интерфейсов взаимодействия между подсистемами, структур данных и самих ППС. Спецификации, на основе которых синтезируются как части (модули, блоки, формы), так и ППС в целом, можно назвать исполняемыми спецификациями.

Кроме вида синтаксиса программ под декларативным программированием еще понимают стиль разработки ПО, при котором при помощи специальной инструментальной среды задаются атрибуты разрабатываемой системы и настраиваются их значения. Как правило, такая инструментальная среда

предоставляет разработчику интерфейс в виде контекстно-зависимых диалоговых форм, встроенные интерпретатор или компилятор, а также среду программирования на императивном языке (например, VB в MS Access, Java в Oracle ADF) для реализации специфических задач.

1.4. Методы интеграции ГИС-функциональности в разрабатываемые системы

Интеграция существующих ППС и ГИС – задача трудоёмкая, требующая для своего решения обязательного наличия исходных текстов информационных систем, освоения разработчиками внедренных ППС геоинформационных технологий, или, наоборот, изучения специалистами в области ГИС исходных текстов уже существующих систем.

Одним из подходов к решению рассматриваемой задачи – это разработка приложений на базе популярных ГИС, таких как ArcView [72], MapInfo [82], MicroStation [84]. В этих системах предусмотрена работа с таблицами БД наряду с семантическими таблицами, связанными со слоями карт, существуют возможности построения запросов, связывающих между собой несколько таблиц и картографических слоев. Однако воспользоваться этими возможностями может лишь опытный пользователь рассматриваемых ГИС, и так как уровень автоматизации этих функций практически отсутствует, то для их реализации необходимо выполнить достаточно большое количество операций даже для привязки к карте небольшого набора таблиц БД. Таким образом, имеющиеся в рассматриваемых ГИС возможности по привязке данных из БД к цифровой карте не позволяют решить задачу интеграции данных в полном объеме, без разработки специализированных приложений, работающих под управлением соответствующих ГИС. С другой стороны, данные программные средства для построения ГИС вынуждают разработчиков использовать внутренние языки программирования (Avenue для ArcView, MapBasic для MapInfo, MDL для MicroStation). Помимо естественной ограниченности выразительных средств

подобных языков этот подход неизбежно порождает проблему кадров и, как следствие, проблему сопровождения программных продуктов.

Учитывая вышесказанное, можно сделать вывод, что при использовании возможностей существующих ГИС задача интеграции данных из БД уже созданных систем, оказывается очень трудоемкой и требует узкой специализации программистов.

1.5. Сравнение инструментальных средств для задачи разработки приложений баз данных

Рассмотрим возможности инструментальных средств Embarcadero Delphi, MS Access и Oracle ADF для решения задачи создания ПБД.

Embarcadero Delphi – является популярным в России представителем интегрированных сред разработки программного обеспечения. Delphi ориентирована на императивную парадигму программирования, при этом (также, как и Embarcadero C++ Builder и Embarcadero RAD Studio) включает в свой состав большой набор VCL-компонентов (Visual Component Library) [45, 90], позволяющих значительно автоматизировать создание ПО за счет реализованных в них алгоритмов и готовых объектов визуального пользовательского интерфейса.

Процесс создания программ в Delphi для работы с БД предполагает реализацию четырех составляющих: механизма доступа к данным, способа представления данных, пользовательского интерфейса для работы с данными, бизнес-логики для обработки данных.

Механизм доступа к данным обеспечивает взаимодействие с источником данных: соединение с БД, доступ и отправку данных. В Delphi в зависимости от выбранной технологии механизм доступа к данным может быть реализован через VCL компоненты TADOConnection (для технологии ADO – ActiveX Data Objects [70]) или TDatabase (для технологии BDE – Borland Database Engine). Данные объекты инкапсулируют методы и свойства, позволяющие управлять доступом к БД в заданной технологии.

Информацию из БД представляют в виде наборов данных – групп записей из таблиц БД, переданных в ПБД для просмотра и редактирования. Каждому набору данных указывается специальный невизульный компонент, обеспечивающий доступ и представление данных. В VCL Delphi реализован набор базовых классов, поддерживающих функции взаимодействия с наборами данных, а также ряд дочерних компонентов для различных технологий доступа к данным (например, TTable, TADOTable, TQuery, TADOQuery с общим предком — классом TDataSet). Компоненты, реализующие взаимодействие с БД через запросы в Delphi, четко разделены на компоненты для получения наборов данных (обрабатывающие SQL-запросы) и модификации данных (выполняющие DML-запросы).

Наборы данных типа TTable, TADOTable обеспечивают доступ и модификацию как к конкретной таблице, так и к нескольким связанным таблицам БД. Эти компоненты позволяют обеспечить доступ как с обычными полями таблиц БД, так и создавать вычисляемые поля. Для выбора данных из подчиненных таблиц существует возможность создания полей-подстановок (Lookup-полей), реализующих возможность выбора записей из подчиненных таблиц, но только на один уровень подчиненности. Необходимо отметить, что такая настройка требуется для каждой таблицы, которая задействована в работе ПБД.

Для реализации пользовательского интерфейса, обеспечивающего взаимодействие с БД, в Delphi реализован ряд визуальных компонентов отображения данных. В основном это аналоги классических визуальных компонентов (TEdit, TGrid), модифицированных для работы с наборами данных (TDBEdit, TDBGrid). Например, TDBGrid обеспечивает отображение набора данных в виде таблицы, а TDBEdit служит для визуализации и редактирования значения заданного поля таблицы. Для работы с полями-подстановками в Delphi существуют специализированные компоненты – TDBLookupListBox, TDBLookupComboBox. Эти компоненты обеспечивают выбор значений из подчиненных таблиц, но не позволяют создавать новые записи. Связь

компонентов визуализации с наборами данных осуществляется через специальный компонент TDataSurce, который отвечает за передачу данных в визуальные компоненты и возврат результатов редактирования в набор данных. TDataSurce контролирует изменение состояния визуальных компонентов при изменении состояния набора данных, передает сигналы управления от пользователя (визуальных компонентов) в набор данных.

При разработке пользовательского интерфейса одной из важных задач является расстановка визуальных компонентов на форме. При этом, с одной стороны, необходимо эффективно использовать пространство формы, а с другой стороны, создать удобный для работы пользователя интерфейс. В Delphi реализован механизм визуального проектирования экранных форм, позволяющий программисту точно задавать расположение каждого компонента. Однако в большинстве приложений пользователю оставляют возможность изменения размеров формы. При этом возникает необходимость в определении правил, контролирующих перемещение компонентов при изменении размеров формы. В роли таких правил в Delphi выступают свойства компонентов Align (выравнивание) и Anchors (якоря).

Свойство Align задает размещение компонента относительно того компонента, на котором он размещен (своего «родителя»). Это свойство может принимать значения alNone, alTop, alBottom, alLeft, alRight, alClient. При значении alNone размер и положение компонента относительно его родителя не изменяются при изменении размеров родителя. При следующих четырех значениях компонент располагается вдоль соответствующей стороны родителя, при этом один из его размеров (вдоль стороны) изменяется, а другой остается постоянным. Значение alClient означает, что элемент управления занимает всю клиентскую область родительского окна.

Свойство Anchors представляет собой набор флагов (akTop, akLeft, akRight, akBottom). Установка каждого из флагов обеспечивает фиксацию элемента управления относительно соответствующей стороны родителя. При задании «якоря» по любой стороне расстояние между соответствующими сторонами

элемента управления и его родителя сохраняется неизменным. Свойство AutoSize обеспечивает изменение размеров компонента в соответствии с размерами его содержимого (текста, изображения, списка, иерархического дерева и т. д.).

Развитые средства визуального программирования и большой набор VCL-компонентов в IDE Embarcadero Delphi позволяют сократить сроки разработки ПО, но все же данная IDE рассчитана на опытных профессиональных программистов. Например, реализация бизнес-процессов, даже если они однотипные, требует написания программного кода для каждой автоматизируемой сущности. При реализации пользовательского интерфейса механизмов перемещения компонентов оказывается достаточно для большинства приложений, но ни один из реализованных в Embarcadero Delphi механизмов не позволяет, например, разделить приращение размера родительского компонента между несколькими его потомками, расположенными в одной строке. Таким образом, создание гибкого динамического интерфейса требует дополнительного кодирования алгоритмов вычисления координат компонентов пользовательского интерфейса и реакции на изменение размеров форм.

Также отметим, что при необходимости модернизации ПБД или интеграции нескольких ППС, реализованных в Delphi, всегда требуется наличие файлов с исходными кодом (*.pas, *.dfm), а их отсутствие приводит к разработке систем «с нуля». Очевидным преимуществом разработки в Delphi в силу универсальности языка программирования является возможность реализации любых необходимых структур и алгоритмов.

Microsoft Access является СУБД, в состав инструментов которой входят средства как управления структурой БД, так и пользовательским интерфейсом. Кроме того, средствами MS Access можно реализовать пользовательский интерфейс для работы с внешними БД, работающими под управлением других СУБД (например, MS SQL Server). Можно сказать, что MS Access – это инструментальное средство для создания сугубо приложений БД.

В состав MS Access входят следующие инструментальные средства: построитель таблиц, построитель экранных форм, построитель SQL-запросов, построитель отчетов, выводимых на печать, построитель макросов, среда разработки VB. Большая часть разработки приложения баз данных в MS Access ведется в декларативном стиле. Например, при создании пользовательских форм и отчетов указываются поля таблицы (нескольких связанных таблиц, запросов) и шаблон, который отвечает за внешний вид (правила расстановки элементов) формы (отчета). Далее сгенерированные формы (отчеты) при необходимости дорабатываются при помощи механизмов визуального программирования в специальных дизайнерах (например, добавляются элементы управления, изменяется оформление). Средства управления положением компонентов на форме аналогичны Delphi. Более сложные функции реализуются программированием на VBA.

Конструктор запросов, входящий в состав MS Access, позволяет визуальными средствами создавать достаточно сложные запросы на выборку записей, а также на модификацию и создание таблиц. Для поддержания возможности формирования сложных условий на поля таблиц в конструкторе запросов встроен построитель выражений. Отметим, что при всей своей мощности конструктор запросов MS Access рассчитан скорее на профессиональных программистов и достаточно сложен для простых пользователей.

Средства автоматизации разработки MS Access позволяют значительно ускорить создание приложений баз данных, но в целом данная инструментальная система имеет ряд ограничений, связанных с встроенной СУБД, и рассчитана на создание ПБД, предназначенных для небольшого количества пользователей, а при взаимодействии с внешними СУБД возникает ряд проблем, которые невозможно решить без доработки ПБД средствами императивного программирования в VB.

Oracle Application Development Framework (Oracle ADF) – современный фреймворк (фреймворк – «каркас», постоянная часть системы, в которой

реализована базовая функциональность [4]), нацеленный на быструю разработку корпоративных ППС. В основе Oracle ADF лежит технология объектно-реляционного отображения с применением развитой MVC (Model-View-Controller [73]) архитектуры. MVC предполагает выделение в приложении трех уровней: модели для взаимодействия с источниками данных, представления для взаимодействия с пользователем (пользовательский интерфейс) и контроллера для управления потоком между моделью и уровнем визуализации. Такое разделение позволяет упростить разработку приложений, поскольку модификация одного из уровней оказывает минимальное воздействие на другие, а сами компоненты одного из уровней (например, компоненты визуального интерфейса) могут многократно использоваться для различных объектов другого уровня.

В Oracle ADF дополнительно выделен слой бизнес-сервисов (Business Services) для доступа к данным из различных источников и реализации бизнес-логики. В качестве источников данных могут выступать реляционные БД, web-сервисы, унаследованные данные. Например, для доступа к реляционным БД используется компонент (сервис) ADF Business Components (ADF BC).

Уровень модели в Oracle ADF (ADF Model) отвечает за взаимодействие уровня пользовательского интерфейса с объектами бизнес-сервисов, обеспечивая единый интерфейс. ADF Model состоит из двух видов компонентов: Data Controls и Data Bindings. Data Controls позволяют абстрагировать детали реализации бизнес-сервисов, а Data Bindings определяют атрибуты, способы управления данными и связывают их с пользовательским интерфейсом, обеспечивая разделение уровня модели и пользовательского интерфейса. Созданная модель приложения хранится в метаданных проекта в виде набора xml файлов.

Уровень визуализации в Oracle ADF может быть реализован как для настольных, так и для web- и мобильных платформ в зависимости от используемых технологий. Так, для реализации настольных приложений можно использовать ADF Swing, AWT-компоненты или интегрироваться в MS Office (создать надстройку в MS Excel). Для реализации web-интерфейса

поддерживаются технологии Java Server Pages (JSP), Java Server Faces (JSF), ADF Faces.

Рассмотрим реализацию настольного приложения БД в Oracle ADF. Значительная часть приложения в Oracle ADF создается в декларативном стиле (через набор диалогов). Вначале необходимо создать соединение с целевой БД, создать и настроить объекты ADF Business Components. Для приложения БД могут быть созданы следующие ADF BC: Сущности (Entity Objects), Представления (View Objects), Ассоциации (Association), Ссылки представлений (Viewlink), Модули приложений (Application Modules).

Сущности соответствуют таблицам БД и содержат структурную информацию о них (типы атрибутов, ключи, ограничения), а также отвечают за выполнение CRUD-операций (CRUD – Create Read Update Delete). Представления – это наборы данных, связанные с сущностями. В простейшем случае представлением может быть подмножество атрибутов определенной сущности или SQL-запросы, формирующие атрибуты из нескольких сущностей в один набор данных. Ассоциации описывают связи между сущностями по первичным и внешним ключам. Ссылки представлений описывают механизм работы пользователя с сущностями и представлениями (например, реализация связи мастер-детали).

Необходимо отметить, что сущности и ассоциации формируются автоматически по выбранным таблицам БД на основе метаданных СУБД. При использовании метаинформации о сущностях и ассоциациях формируются представления и ссылки представлений. Кроме того, представления можно объединять в один или несколько модулей приложений. Модуль приложения создает и управляет транзакциями базы данных, а также для слоя модели предоставляет данные и методы для доступа.

В результате настройки ADF BC будут сгенерированы Модель Базы Данных и Контроллеры данных. В Модели БД автоматически создается Модуль приложения БД, запустив который можно оценить работоспособность построенной модели.

При разработке слоя визуализации в Oracle ADF необходимо создавать формы для каждого представления (созданных на этапе формирования модели БД). Для автоматизации создания форм существуют два подхода. Первый подход – настройка формы в диалоговом режиме с последующей генерацией. При этом можно указать тип формы (одна таблица или запись с таблицей деталей), шаблоны компоновки элементов на форме (поля записи можно выстроить в один или два столбца и указать место подписей). Второй подход – создание пустой формы и перетаскивание на нее контролеров данных, в результате чего на форме автоматически генерируются панели с полями представлений. Кроме того, формы для работы с представлениями можно создавать в ручном режиме (размещение элементов интерфейса с последующей привязкой к источникам данных).

Средства автоматизации Oracle ADF позволяют ускорить разработку информационных систем, но в целом данная технология дает преимущества при разработке крупных ППС и рассчитана на большой коллектив разработчиков, для которых модель ППС будет обеспечивать согласованное взаимодействие. Необходимо отметить, что средства автоматизации Oracle ADF не избавляют в полном объеме разработчиков от необходимости написания программного кода.

Таблица 1. Сравнение инструментальных средств

	Delphi	MS Access	Oracle ADF
Тип инструментального средства	IDE	РСУБД	Framework
Парадигма разработки	Императивная, сборочная	Декларативная	Смешанная (декларативная + императивная)
Средства автоматизации разработки	VCL	Визуальные диалоги и конструкторы	Визуальные диалоги и генерация кода
Наличие модели	Нет	Да	Да

разрабатываемого приложения БД			
Средства представления модели разрабатываемого приложения БД	–	Скрыто	Репозиторий метаинформации, XML-файлы
Создание пользовательских форм	«Ручная» сборка из визуальных компонентов.	Генерация по шаблонам с возможностью «ручной» доработки	Генерация по модели, «ручная» сборка из компонентов
Средства компоновки пользовательских форм	Управление свойствами визуальных компонентов (выравнивание и якорь)	Шаблоны, управление свойствами визуальных компонентов.	Шаблоны, менеджеры компоновки.
Сложность модификации при изменении структур БД (например, добавление новых полей)	Только при наличии исходного кода. Требуется перенастройка компонентов взаимодействия с источниками данных и модификация форм.	Требуется перенастройка таблиц, генерация или модификация форм в «ручном» режиме	Только при наличии каркаса приложения. Требует модификации свойств ВС, модели и генерация или «ручная» сборка форм.

Реализация ГИС-функциональности в готовом приложении.	Только при наличии исходного кода. Разработка и реализация картографического модуля средствами (API) популярных ГИС.	Разработка и реализация картографического модуля средствами (API) популярных ГИС.	Только при наличии каркаса приложения. Разработка и реализация картографического модуля средствами (API) популярных ГИС.

Специализированные средства (например, My Visual DataBase Drive Software company, Corel Paradox, dBase Plus 8) существенно уступают по своей функциональности вышеперечисленным средствам общего назначения, в силу своей ориентированности на оптимизацию решения отдельных информационных задач.

Выводы

Современные методы разработки прикладных программных систем ориентированы на тесное взаимодействие с пользователями заказчика, что требует применение технологий, позволяющих быстрое создание прототипов ППС, их представление, внесение корректировок и доработку. Существующие технологии разработки ПБД не предоставляют (либо не поддерживают) возможность использования моделей с требуемым уровнем детализации представления знаний о связях используемых структур данных с их возможным отображением в пользовательский интерфейс. Средства автоматизации реализации пользовательских интерфейсов имеют невысокий уровень и недостаточную гибкость к изменениям в структуре системы (модификации и масштабированию).

Таким образом, возникает необходимость в разработке новой модели, языка и инструментальных средств позволяющих повысить эффективность разработки ПБД, обладающих функциями взаимодействия с внешними ППС и с пространственными данными, а также имеющих возможность гибко масштабироваться на протяжении всего жизненного цикла.

Глава 2. Технология автоматизации создания приложений БД

2.1. Предлагаемый подход

Для повышения скорости создания и легкости модернизации ПБД в работе предлагается использовать подход к разработке, ориентированный на модель, содержащую достаточную информацию для автоматического создания пользовательских интерфейсов приложений БД, обеспечивающих поддержку выполнения CRUD-функций, построения запросов, взаимодействия с ПД, а также возможность решения специфических задач.

В качестве средства хранения и представления моделей приложений БД предлагается использовать декларативные спецификации, написанные на специально разработанном языке спецификаций. Конструкции данного языка обеспечивают достаточно детальное и в то же время компактное описание всех элементов приложений БД в рамках предлагаемой технологии. Кроме того, представление моделей в виде спецификаций позволяет поддержать модульную разработку приложений БД – интегрировать готовые спецификации приложений при разработке новых.

Для автоматизации создания спецификаций предлагается использовать инструментальное систему, обеспечивающую интерактивное формирование всех элементов моделей приложений БД. Готовое предметное ПБД создается динамически в результате интерпретации спецификации в той же инструментальной системе.

Рассмотрим предлагаемую технологию разработки приложения БД. Процесс разработки ПБД предлагается разбить на следующие этапы (Рисунок **1**):

1. Структурный анализ предметной области.
2. Проектирование БД.
3. Создание модели ПБД при помощи инструментального средства «ГеоАРМ» [8, 11, 12, 15, 20, 51, 55, 56].

4. Автоматическое создание ПБД, обладающего ГИС-функциональностью, на основе спецификации инструментальной системой «ГеоАРМ».

Рисунок 1. Этапы технологии разработки ПБД

На первом этапе ИТ-специалистом проводится структурный анализ предметной области, в результате которого выявляются сущности, формулируются требования к структуре, описываются бизнес-процессы, выявляются функциональные зависимости. Далее следует этап проектирования БД, результатом которого является схема БД для разрабатываемой системы, реализованная в определенной СУБД. Для автоматизации этого этапа можно использовать уже зарекомендовавшее себя ПО от ведущих компаний-разработчиков, позволяющее строить ER-модели данных и генерировать схемы БД (например, интегрированные утилиты СУБД MS SQL Server, Oracle или CASE-средства такие, как Sybase Power Designer [88], IBM Rational Rose [78]).

На третьем этапе строится модель приложения БД [49, 53, 55, 57, 58]. При этом в качестве входных данных используется метаданные о структуре БД (схема БД) хранящиеся в СУБД. Схема данных является уже структурированными знаниями о сущностях и связях между ними, но этих знаний недостаточно для создания полноценного ПБД. Структурную информацию БД необходимо расширить знаниями о бизнес-процессах и способах представления информации пользователю разрабатываемого ПБД.

В процессе создания модели ПБД разработчику необходимо сформировать следующие объекты: «Таблицы», «Представления», «Правила», «Надстройки» (Рисунок **2**). «Таблицы» содержат структурную информацию, обеспечивающую взаимодействие (выполнение CRUD-операций) с соответствующими объектами (таблицами) БД, а именно информацию о полях таблиц с их типами данных и связях с другими таблицами БД. В случае грамотно спроектированной БД структурные метаданные таблиц можно получить автоматически из СУБД. Но необходимо отметить, что встречаются БД, в которых отсутствует информация о связях (например, унаследованные БД), поэтому должна быть возможность уточнения такой информации при формировании объектов модели приложения БД. Та же возможность необходима при реализации работы в приложении с несколькими БД.

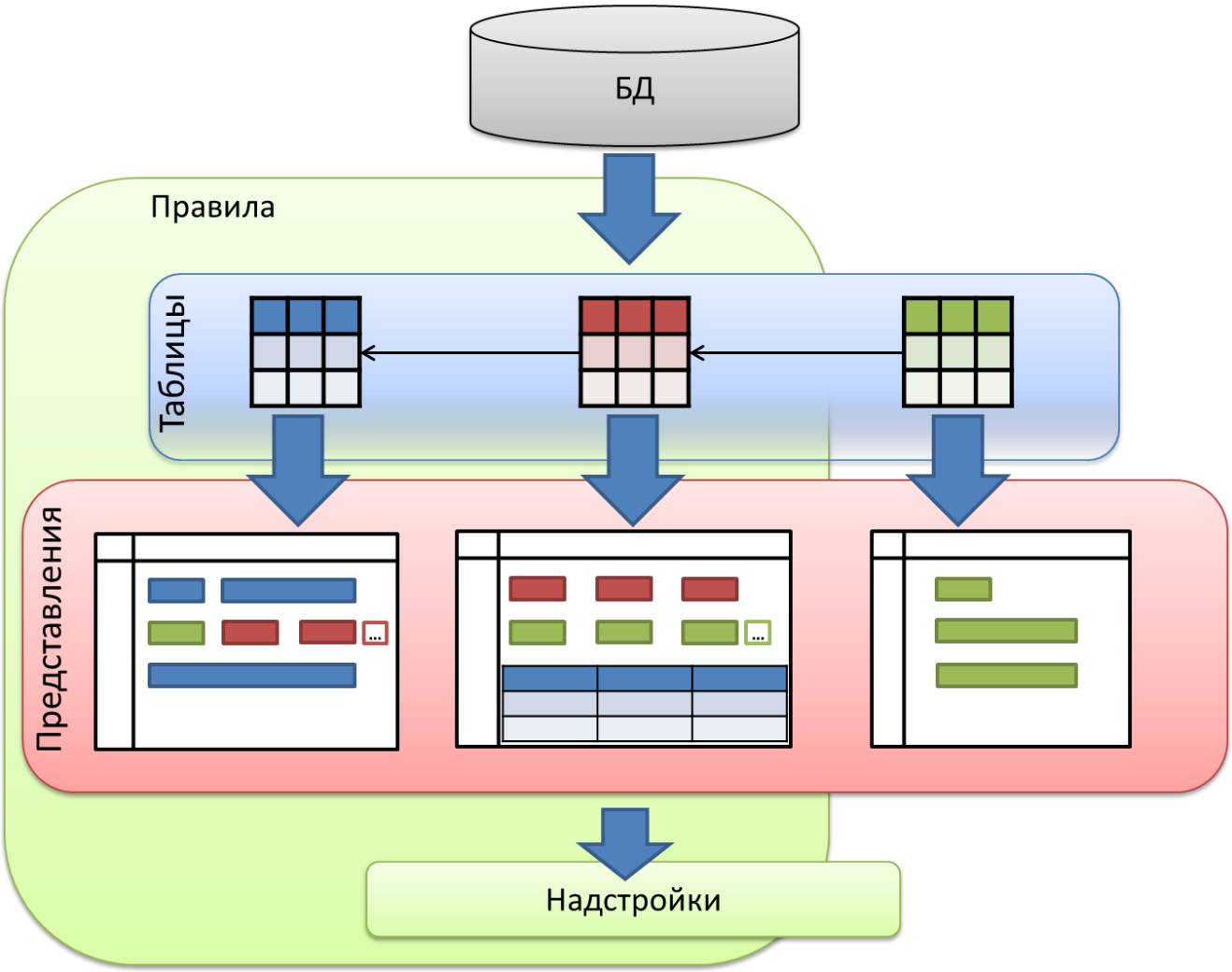

Рисунок 2. Объекты модели приложения БД

«Представления» – это более сложные, чем «Таблицы», объекты модели ПБД, отвечающие за формирование наборов данных для последующего отображения. «Представления» необходимы для формирования пользовательского интерфейса, поскольку только структурной информации о таблицах БД для этого недостаточно. Например, при наличии в таблице внешнего ключа (ссылка на справочник) пользователь должен иметь возможность выбирать значения из справочника, а если справочник сложный (состоит из нескольких полей или содержит в свою очередь ссылки на другие справочники), то необходимо знать, какая комбинация полей справочника позволяет однозначно идентифицировать нужную запись (по каким полям выбирать запись) и есть ли необходимость при этом отображать значения через несколько уровней ссылок. Кроме того, структурной информации о связях бывает недостаточно для

понимания желаемого механизма работы пользователя с данными. Например, ссылка из списка А в справочник В (внешний ключ) при работе с записями списка А обеспечивает выбор значений из В, а для каждой записи В позволяет формировать список значений из А (механизм «Мастер-детали»). В описании «Представлений» содержится информация о декодировании значений по ссылкам и способах интерпретации связей, что обеспечивает автоматическое построение удобного пользовательского интерфейса.

Для большинства ПБД, предназначенных для решения сугубо реестровых задач (например, регистр населения, реестр объектов недвижимости), при создании приложения БД достаточно описать «Таблицы» и «Представления». Объекты модели «Надстройки» необходимы для поддержки взаимодействия с внешними модулями, отвечающими за решение не заложенных в ПБД задач (например, сложные вычисления, построение графиков).

Объекты модели «Правила» отвечают за реализацию специфических бизнес-функций между различными объектами модели ПБД, например, установка режимов модификации данных, проверка на уникальность, взаимодействие с ПД. Правила могут влиять как на работу с конкретными таблицами БД, так и на формирование пользовательского интерфейса. Например, при наличии в таблице координат пространственного объекта формируются специальные элементы пользовательского интерфейса для поддержки функций связи с цифровыми картами и автоматического построения пространственных объектов.

Для поддержки процесса создания моделей ПБД применяется специально разработанная инструментальная система «ГеоАРМ» [8, 11, 12, 15, 20, 51, 55, 56]. Инструментальная система обеспечивает поддержку следующих процессов: подключение к БД, загрузка структурных метаданных таблиц БД, уточнение структуры и связей таблиц, формирование представлений, задание правил, описание взаимодействия с надстройками, а также уточнение механизмов отображения (для формирования пользовательского интерфейса). Необходимо отметить, что такое уточнение не является необходимым, поскольку структурной информации о таблицах и представлениях в предлагаемой технологии достаточно

для создания работоспособного ПБД. В результате созданные модели ПБД инструментальная система позволяет сохранить в виде декларативных спецификаций [15, 47, 51, 55].

Часто в информационных технологиях спецификации являются описанием архитектуры и функций ППС, создаваемые для согласования действий разработчиков и тестирования прототипов. В предлагаемом подходе спецификация является средством представления и хранения модели ПБД, которая в декларативном виде содержит информацию о предметной области в виде соответствующих структурных метаданных БД, правилах формирования пользовательского интерфейса, бизнес-логике приложения, методах взаимодействия с ПД и способах взаимодействия с внешними приложениями.

На четвёртом этапе при помощи спецификации инструментальная система автоматически настраивается, становясь предметным ПБД, обеспечивающем поддержку всех функций работы с базой данных и возможность взаимодействия с пространственной информацией. Весь пользовательский интерфейс ПБД формируется динамически при интерпретации спецификации системой «ГеоАРМ» (Рисунок **3**). Преимущества такого подхода состоит в отсутствии необходимости специально разрабатывать (создавать элементы интерфейса, связывать их с данными, компилировать, отлаживать) формы пользовательского интерфейса для каждой таблицы и представления. Механизм создания элементов пользовательского интерфейса, связывания их со структурами данных унифицирован и встроен в интерпретатор системы. Интеграция функций создания спецификаций и их интерпретации в одной системе даёт возможность разработчику сразу оценивать адекватность создаваемой модели ПБД и позволяет повысить скорость разработки в целом.

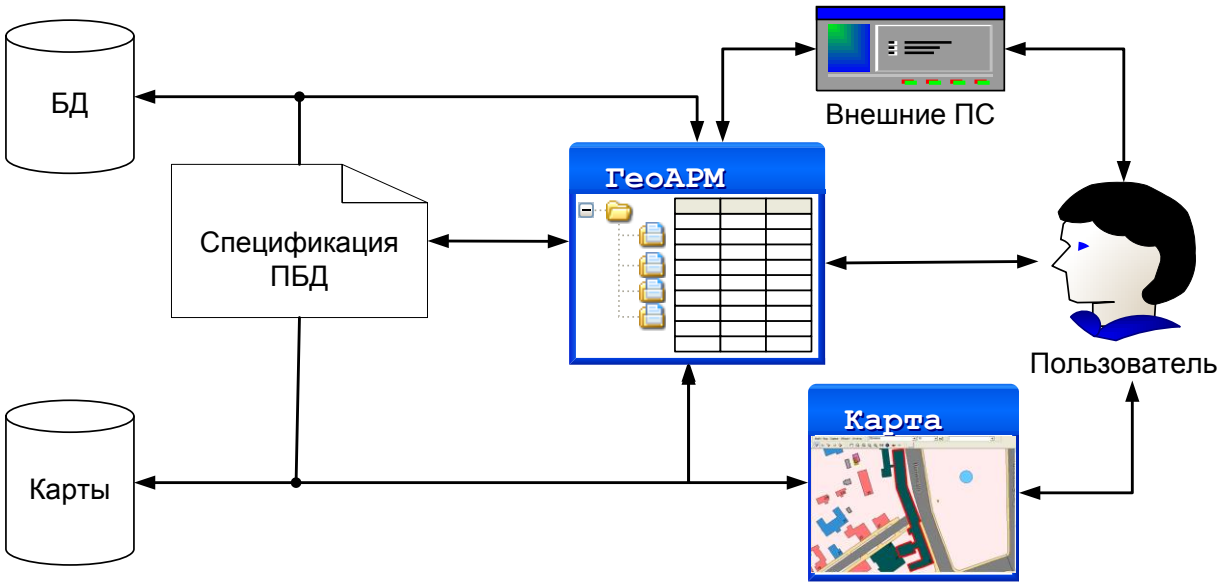

Рисунок 3 Инструментальная система «ГеоАРМ»

Развитие и модернизация ПБД в предлагаемой технологии упрощается за счет отсутствия потребности в изменении программного кода, его отладки и компиляции с применением специализированных средств разработки ПО. Разработанная в рамках данной работы инструментальная система обеспечивает поддержку как создания, так и модернизации спецификации ПБД за счет встроенных визуальных инструментальных средств формирования спецификаций, что позволяет сократить сроки доработки ПБД на каждом этапе жизненного цикла системы.

2.2. Модель приложения БД

Для автоматизации создания приложений БД предложена модель, обеспечивающая отображение схемы БД в пользовательский интерфейс. В модели приложения БД выделены следующие структуры:

Schema – структура, описывающая схему БД (таблицы), расширенную информацией о связях, способах организации данных для пользователя (представления) и связях с пространственными данными;

Display – структура, описывающая правила отображения данных пользователю, формирования пользовательского интерфейса;

Rules – структура для описания специфических бизнес-правил приложения;

Plugins – структура для описания взаимодействия с внешними подключаемыми ППС, решающими задачи не заложенными в базовый функционал.

Таким образом, модель приложения БД можно представить в следующем виде:

M=<Schema,Display,Rules,Plugins>

Рассмотри структуры модели более подробно. Структура *Schema=<Tbls,Refs>* описывает множества таблиц $Tbls=\{t_1,\dots,t_n\}$, $n \in N$ и связей $Refs=\{r_1,\dots,r_m\}$, $m \in N$ между ними. Для всех таблиц БД определено множество схем $C=\{c_1,\dots,c_l\}$. Тогда $t_i(c_j)$, $i=1,\dots,n$, $j=1,\dots,m$ – таблица со схемой c_j, где $c_j=\{a_1,\dots,a_p\}$ схема таблицы t_i, а a_h, $h=1,\dots,p$ – атрибут таблицы.

Каждый атрибут относится к определённому типу данных: $\forall a_h \; \exists Type(a_h)$ – тип атрибута. Определенны восемь типов $Type=\{I,F,S,D,B,X,G,SD\}$, где I – множество целых чисел, F – множество действительных чисел, S – множество строковых значений, D – множество значений дат и времени, $B=\{true;false\}$ – булевы значения, X – множество бинарных данных, G – множество растровых данных, SD – множество пространственных данных. Тип атрибута таблицы влияет на способ отображения и обработки данных из этого поля.

Каждая таблица состоит из конечного множества записей $t_i(c_j)=\{k_1,\dots,k_q\}$, $q \in N : \forall k_g \; c_j \rightarrow Type`$, $k_g(a_h) \in Type_f$, $f=1,\dots,8$, $g=1,\dots,q$; где $k_g(c_j)$ – запись со схемой c_j, а $k_g(a_h)$ – значение записи k_g на атрибуте a_h, т.е. значение поля.

Для выявления связей между таблицами должны быть определенны ключи. Первичный ключ $\boldsymbol{PK}=\{a_1,\dots,a_{pp}\} \subseteq c_j$, $pp=1,\dots,p$, если

1. $\forall k_g, k_{gg} \in t_i: k_g(\boldsymbol{PK}) \neq k_{gg}(\boldsymbol{PK})$, $g \neq gg$, $g=1,\dots,q$, $gg=1,\dots,q$;
2. $\exists \boldsymbol{PK}' \subseteq \boldsymbol{PK}: k_g(\boldsymbol{PK}') \neq k_{gg}(\boldsymbol{PK})$, $g \neq gg$, $g=1,\dots,q$, $gg=1,\dots,q$.

Часто в качестве ***PK*** используют так называемые искусственные ключи – автоинкрементные поля, например $k_{g+1}(\boldsymbol{PK}) = k_g(\boldsymbol{PK})+1$. При этом в таблице могут присутствовать естественные ключи (например, «Район», «Улица», «Дом»). Комбинации полей естественных ключей $\boldsymbol{NF}=\{a_1,\dots,a_{ppp}\} \subseteq c_j$, $ppp=1,p$ будем

называть именующими полями. Такие именующие поля необходимы для организации удобного пользовательского интерфейса, например, по ним может быть организована автофильтрация (autocomplete) или сортировка.

$\boldsymbol{FK}=\{a_1,\ldots,a_{pp}\}\subseteq c_{jj}$ – внешний ключ таблицы t_{ii}, $jj=1,\ldots,m$, $ii=1,\ldots,n$, если

1. $\exists\, t_i(c_j)$, $\boldsymbol{PK}=\{a_1,\ldots,a_p\}\subseteq c_j$;
2. $\forall\, k_{gg}\in t_{ii}(c_{jj})\ \exists\, k_g\in t_i(c_j) : k_{gg}(\boldsymbol{FK}) = k_g(\boldsymbol{PK})$, $gg=1,\ldots,q$.

Структура $Refs=\{r_1,\ldots,r_m\}$ описывает множество связей между таблицами. Данная структура необходима для автоматического создания наборов данных для взаимодействующих с пользовательским интерфейсом, поскольку метаинформации СУБД о ключах недостаточно для понимания, как должна быть организована работа с таблицами с точки зрения пользователя. В модели определены связи трёх видов: $r_f\in\{LR,PR,DR\}$, $f=1,\ldots,m$.

Связи вида *LR* будем называть простыми связями или ссылками на справочники.

$r_f=LR(t_i, t_{ii})$, *если* $k_g(\boldsymbol{FK}) = k_{gg}(\boldsymbol{PK}) \vee \varnothing$, *где* $\boldsymbol{FK}\in t_i$, $\boldsymbol{PK}\in t_{ii}$, $k_g\in t_i$, $k_{gg}\in t_{ii}$.

Связи вида *PR* – это связи по первичным ключам.

$r_f=PR(t_i, t_{ii})$, *если* $k_g(\boldsymbol{PK})=k_{gg}(\boldsymbol{PK})$, *где* $\boldsymbol{PK}\in t_i$, $\boldsymbol{PK}\in t_{ii}$, $k_g\in t_i$, $k_{gg}\in t_{ii}$.

Такие связи необходимо отличать от ссылок на справочники поскольку бизнес-правила совместной работы с таблицами отличаются. Например, при работе с представлением построенном на таблице **A** связанной с таблицей **B** связью вида *PR*, сначала создаётся запись в таблице **A**, а соответствующая запись (с таким же **PK**) в таблице **B** должна создаваться только при редактировании её полей (таблицы **B**).

Связи вида *DR* – связь типа детали.

$r_f=DR(t_i, t_{ii})$, *если* $k_{gg}(\boldsymbol{PK})=k_1(\boldsymbol{FK})\wedge\ldots\wedge k_g(\boldsymbol{FK})$, *где* $\boldsymbol{FK}\in t_i$, $\boldsymbol{PK}\in t_{ii}$, $k_{gg}\in t_{ii}$, $\{k_1,\ldots,k_g\}\in t_i$.

Информация о наличии таких связей необходима для формирования пользовательского интерфейса, обеспечивающего работу с деталями для мастер записи. По сути, связь *DR* это интерпретация связи *LR* со стороны справочника, показывающая, что для каждой записи справочника необходимо отобразить набор

записей из таблицы списка, содержащих ссылки на текущую запись справочника. Например, представление зданий расположенных на определённой улице, когда улицы являются частью адреса зданий.

Наличие связей типа *LR* или *PR* позволяют строить представления. Представление (Рисунок 4) $v(c') \in V$ это набор данных построенный на таблице $t_i(c_j) \in Tbls$, где $c' \subseteq c_j$ (содержит подмножество атрибутов таблицы $t_i(c_j)$), либо $v(c')=(t_i(c_j), V', LR', PR')$, где $V' \subseteq V$, $\forall v \in V' \exists lr(t_i,v) \in LR' \vee \exists pr(t_i,v) \in PR' \wedge c' \subseteq c_j \cup c'_1 \cup \ldots \cup c'_l$. В отличии от View популярных СУБД, которые являются сохранёнными SQL-запросами, представления в предлагаемой модели являются наборами данных, которые построены на одной базовой таблице и содержат информацию о декодировании связей для получения данных из связанных таблиц. Такие представления позволяют автоматически организовать работу со связанными таблицами, как с одним набором данных. На Рисунок **4** представление построено на таблице **T1**и содержит значения из таблиц **T2** и **T3** полученные через декодирование связей типа *PR* с **T3** и *LR* с **T1**.

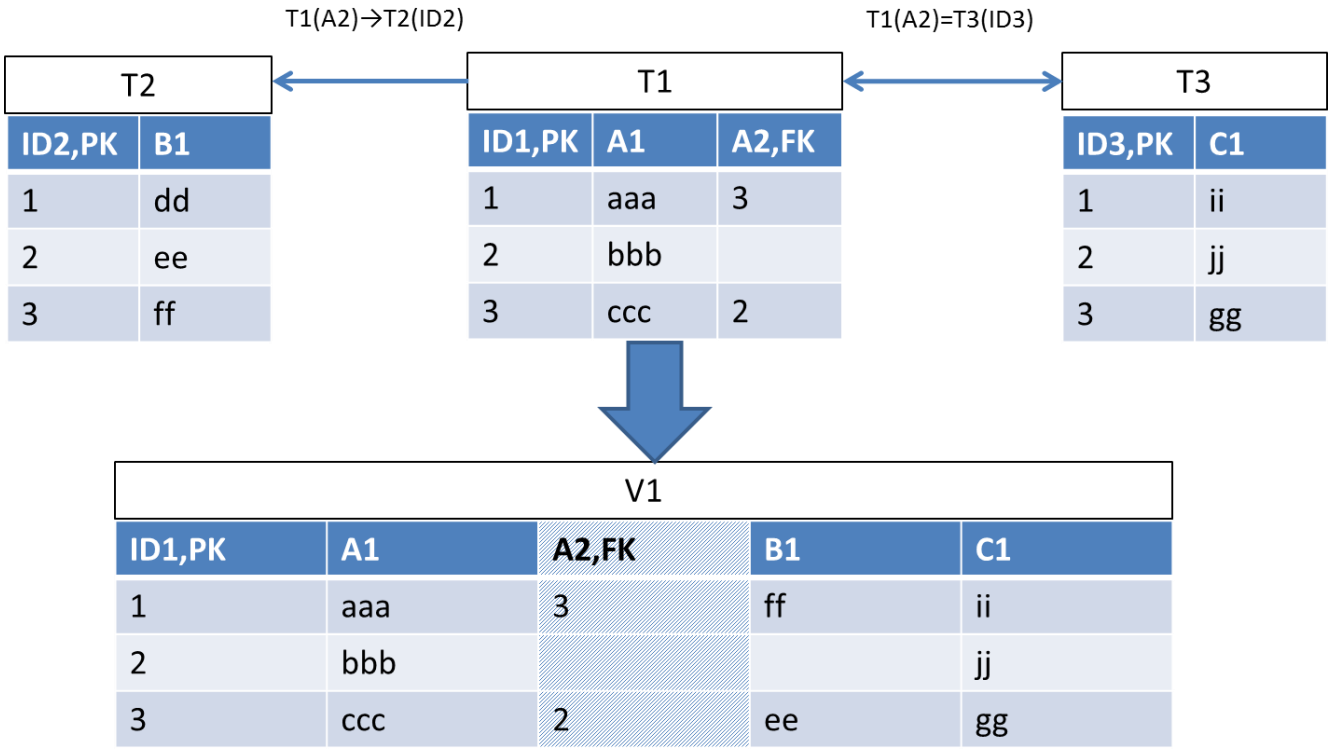

Рисунок 4 Пример представления

Структура ***Display***=<*C,Refs,Mod,E,C→E*>, отвечает за формирование пользовательского интерфейса. В *Display* содержатся унифицированные правила отображения данных из таблиц и представлений в элементы пользовательского интерфейса.

Как правило, современные приложения БД имеют оконный интерфейс. Каждое окно приложения ориентировано на решение определённой задачи и является формой, на которой размещены визуальный компоненты интерфейса. Для ПБД свойственны следующих визуальные компоненты: *Btn* – кнопка, *Edt* – текстовое поле, *Lbl* – надпись, *Cbx* – выпадающий список для выбора значений, *Dtp* – календарь, *Chkt* – флаг, *Img* – поле с картинкой, *Grd* – сетка (таблица), *GB* – контейнер, *TS* – закладка. *Edt, Lbl, Cbx, Dtp, Chkt, Img* – относятся к информационным компонентам, их задача отображение информации пользователю. *Grd, GB, TS* – организационные компоненты, позволяющие структурировать другие компоненты на формах. *Btn* – является элементом инициирования события. Выделим как отдельный компонент *fCbx* – выпадающий список с функцией «автокомплит» (автоматическая фильтрация при вводе значений). И так, элементы пользовательского интерфейса для представления и обработки данных из БД можно представить следующим множеством:

$$E=\{Btn,Edt,Lbl,Cbx,Dtp,Chkt,Img,fCbx,Grd,GB,TS\}.$$

Практических во всех приложениях БД для работы пользователя с каждой таблицей реализуется два режима *Mod=(List,Card).* В режиме *List* данные представляются пользователю в виде списка записей. При этом используется элемент *Grd*. Другой режим *Card* предназначен для работы с индивидуальной записью таблицы (представления) и реализуется через комбинацию визуальных компонентов из *E* размещённых на форме.

Для каждого атрибута $a_h \in c_j$, $h=1,\ldots,p$ на форме должны создаваться элементы идентификатор и модификатор. В качестве идентификаторов выступают элементы *Lbl,* в которые отображаю имена атрибутов. Элементы модификаторы определяются на основе типов атрибутов структуры **Schema** и их принадлежности базовой или связанным таблицам (полученным по ссылкам):

$k(a_h) \rightarrow Edt$, *если* $Type(a_h)=I \vee F \vee S$, $k \in t_i$, $h=1,\ldots,p$;

$k(a_h) \rightarrow Dtp$, *если* $Type(a_h)=D$, $k \in t_i$, $h=1,\ldots,p$;

$k(a_h) \rightarrow Chkt$, *если* $Type(a_h)=B$, $k \in t_i$, $h=1,\ldots,p$;

$k(a_h) \rightarrow Cbx$, *если*, $Type(a_h)=I \vee F \vee S$, $k(a_h) \in v(c')=(t_i(c_j), V', LR', PR')$, $a_h \in c_j$, $h=1,\ldots,p$;

$k(a_h) \rightarrow Img$, *если* $Type(a_h)=G$, $k \in t_i$, $h=1,\ldots,p$;

$k(a_h) \rightarrow fCbx$, *если* $Type(a_h)=I \vee F \vee S$, $k(a_h) \in v(c')=(t_i(c_j), V', LR', PR')$, $a_h \in NF \subseteq c_j$.

В случаях, когда поля получены по ссылкам (в представлениях) или являются **FK** рядом с ними присутствует кнопка (*Btn*) для перехода к записям по ссылке. Несколько полей могут быть выделены в отдельные блоки. Поддерживается два вида блоков – группа (*GB*) и закладка (*TS*). Если $Type(a_h)=SD$, то создаётся кнопка (*Btn*) вызывающая таблицу связей объектов цифровой карты с записями таблиц (представлений).

Структура *Rules={RO,CHK,MO,ROWCOLOR,TEXTCOLOR,FILTER}* отвечает за специфические режимы и правила работы с данными. Например, если $RO(t_i)$, то работа с таблицей t_i будет в режиме «только для чтения», поля таблицы будут не активны для редактирования.

Режим $CHK(t_i(c`_j),\{0,1\})$ отвечает за проверку вводимых данных на уникальность значений, при этом существует два режима проверки: ошибка (1) и предупреждение (0). В первом режиме при вводе уже существующего в таблице значения сохранение записи блокируется, пока пользователь не введёт уникальные значения. Второй режим предупреждает о совпадении вводимых значений с уже существующими, но сохранять позволяет.

Механизм $MO(t_i, t_{ii}, r_f=DR(t_i, t_{ii}), loc_i \in LOC)$ позволяет автоматически создавать пространственные объекты вида $loc_i \in LOC$, *LOC={Point, Line, PolyLine, Polygon, PolyPolygon }* для записей из t_i на цифровой топооснове по метрике из таблицы деталей t_{ii}

Структура *Plugins=<Tbls,D,Dll>* отвечает за взаимодействие с внешними приложениями, решающими специфические задачи. Данная структура обеспечивает для $t_i \in Tbls$ передачу данных во внешнее приложение $dll_z \in Dll$, вызов которых осуществляется из пользовательского интерфейса $d_x \in D$, $D \subseteq Display$.

ROWCOLOR и *TEXTCOLOR* режимы задания цвета и текста в строках таблицы (для элемента интерфейса *Grd)* в зависимости от значений определённых атрибутов.

2.3. Язык спецификаций приложений БД

Для создания спецификаций ПБД разработан предметно-ориентированный язык – ЯПБД (Язык Представления Баз Данных) [15, 57, 60]. Грамматики разработанного языка принадлежат к классу LL(1) [3]. Конструкции языка позволяют в декларативном виде описывать элементы предложенной модели ПБД, а также некоторые общие настройки системы (например, способ подключения к БД). Созданные на ЯПБД спецификации удовлетворяют требованиям *точности, понятности и полноты* [1, 2], т.е. в спецификациях на ЯПБД содержится вся необходимая (в рамках предложенной технологии) информация для решения задачи автоматического создания ПБД, все объекты модели хорошо формализованы, при этом спецификации достаточно компактны и в тоже время понятны (читабельны).

Предложения ЯПБД имеют следующую структуру:

<Стартовое слово> <Список обязательных выражений> [список необязательных выражений]

Начало предложений определяется по принадлежности первого слова к множеству стартовых слов. Каждое предложение состоит из обязательных и необязательных выражений. Все выражения разделены пробелами. Каждое выражение содержит зарезервированное (служебное) слово, описывающее имя какого-либо атрибута системы, и может включать значение этого атрибута через знак «=». Например, **`READONLY`** или **`SCHEMA`**=`<Значение атрибута>`. В некоторых случаях за служебным словом может идти список выражений в

скобках «(…)» через запятую «,»: **FIELDS**(<выражение>,…,<выражение>). Предложения ЯПБД и их назначение описаны в таблице 2[1].

Таблица 2. Предложения ЯПБД.

Предложения ЯПБД	Стартовое слово	Назначение
Подключение к БД	**ADO** **BDE**	Описание параметров подключения к целевой БД в технологиях ActiveX Data Objects (ADO) [70] или Borland Database Engine (BDE).
Настройки	**CFG**	Описание общих настроек приложения БД.
Таблица	**Table**	Описание структуры таблицы в терминах модели приложения БД. Создаётся для каждой таблицы БД, с которой необходимо реализовать взаимодействие.
Представление	**View**	Описание представления в терминах модели приложения БД. Создаётся для работы со связанными таблицами БД.
Надстройки	**Plugins**	Описание механизма взаимодействия с внешними системами.
Связи со спецификациями	**Uses** **Include**	Описание механизма интеграции с другими спецификациями (созданными в этой же технологии).
Главное меню	**Menu**	Описание структуры главного меню приложения БД.

[1] Более подробна семантика ЯПБД описана в Приложении А.

В описании Настроек (**CFG**) задаются общие параметры работы ПБД. Например, отображаемое наименование приложения (**AppTitle**), схема БД (**Schema**) в СУБД, режим работы с БД, способ формирования имён полей и таблиц для запросов.

Предложения, описывающие таблицы и представления сочетают в себе информацию о всех структурах предложенной модели ПБД **M=<*Schema,Display,Rules,Plugins*>**. Например, в описании (Рисунок **5**) поля «DOM=S» указан строковый тип, что при интерпретации обеспечит создание на форме элемента *Edt* (структуры ***Display*** модели) для взаимодействия с этим полем. Наличие информации о ссылке типа *LR (*строка «KOD_UL=^vSP_UL»*)* позволяет построить представление, для которого в автоматически созданной форме будут размещены выпадающие списки – элементы *Cbx*, обеспечивающие выбор значений полученных по ссылке из полей представления vSP_UL, а также кнопка (*Btn*) для перехода по ссылке к форме с записью из vSP_UL. Если поля подстановки в описании целевой таблицы (справочника) указаны как именованные (**NF**), то на форме для них будут созданы *fCbx* (выпадающие списки с функцией автокомплит).

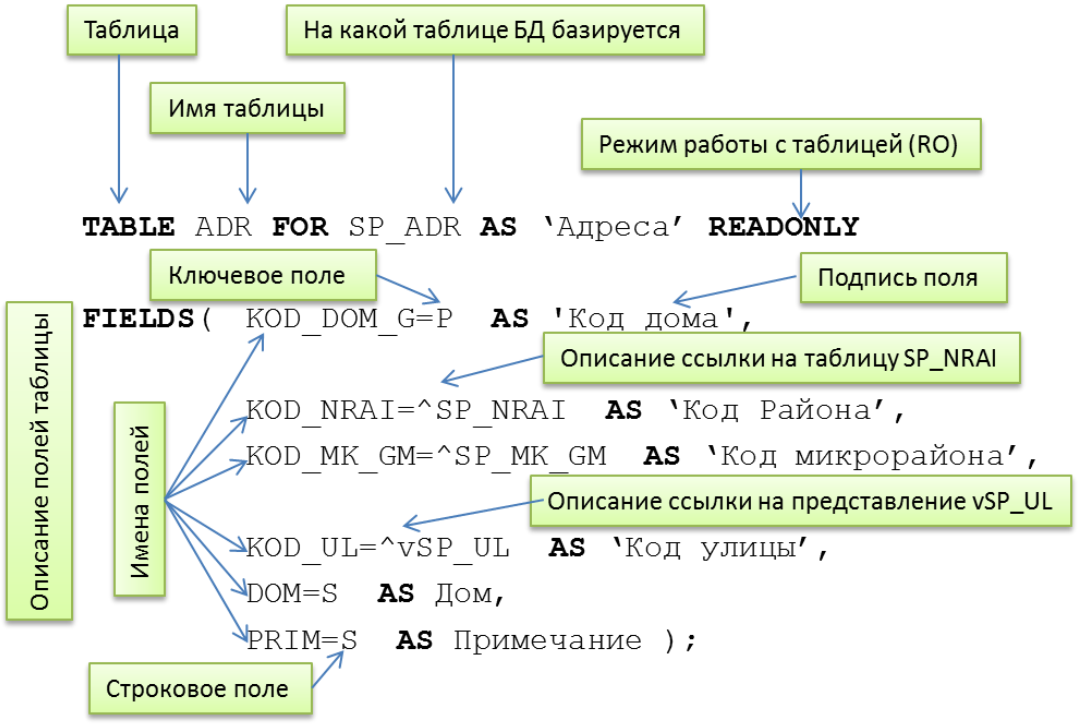

Рисунок 5 Пример описания таблицы на ЯПБД

Дополнительно к возможности указания ссылок в описании полей таблиц предусмотрена конструкция **REFS**(…), позволяющая задавать ссылки любых типов, в том числе и «Мастер-детали» (**DR**). Например, наличие в описании таблицы **SP_UL** конструкции **REFS**('Здания улицы'=<vSP_ADR(KOD_UL)) указывает на необходимость при работе с записями таблицы SP_UL (или представления на основе этой таблицы) отображать список записей (детали) из представления **vSP_ADR**, у которых в поле KOD_UL содержится значение **PK** текущей записи **SP_UL.** При этом в списке деталей будут выводится все поля представления **vSP_ADR** за исключением полей получаемых из **SP_UL.**

Наличие для таблиц атрибута **READONLY** блокирует возможность редактирования полей в создаваемых формах как для таблиц, так и для основанных на них представлениях. При этом все значения полей на форме отображаются без возможности редактирования в элементах *Edt.*

В описании представлений (Рисунок **6**) указывается базовая таблица, являющаяся основой этого представления, необходимые поля из базовой таблицы и поля подстановки, значения которых получаются через декодирование ключей ссылок (например, R=KOD_NRAI.RAION). Возможность описывать ссылки на представления позволяет отображать значения подстановочных полей этих представлений. При этом происходит декодирование цепочки ключей для получения значений через несколько уровней ссылок.

Механизм взаимодействия ПБД с цифровой картой описывается набором атрибутов задающих поля таблицы (**MAPFLD**=) содержащие связи с ПД, таблицу связей (**MAPLNKTBL=**) и номер цифровой карты (**MAPKIND=**). Кроме того, могут быть указаны поля таблицы, которые содержат адрес (Улица, Дом) объекта (**ADDRS**='UL;DOM'), что обеспечит связь с объектами цифровой карты через геокодирование.

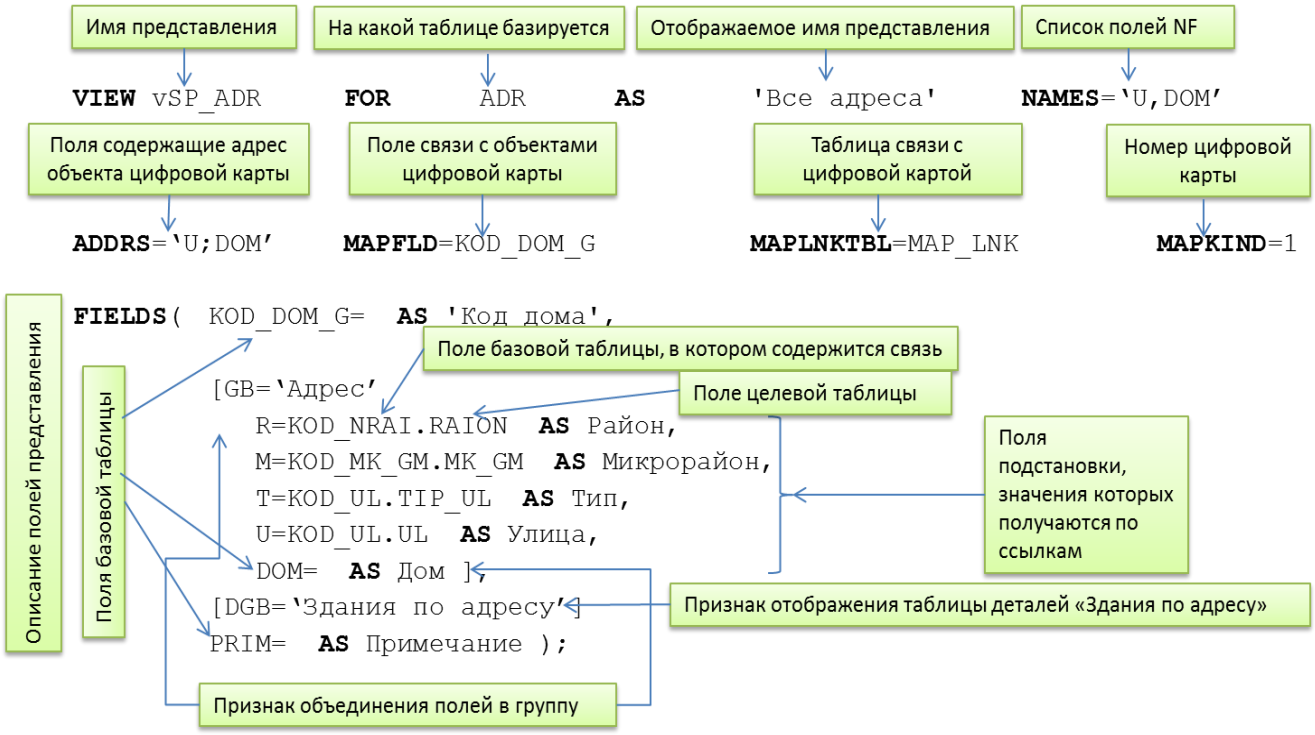

Рисунок 6 Пример описания представления на ЯПБД

Интеграция с другими спецификациями приложений БД может быть описана двумя способами. Конструкция **USES** позволяет использовать уже описанные в указанных спецификациях структуры (предложения). Например, **USES** ADRS=('addrs.dbpl', 'AnotherDB.dbo.', READONLY) позволяет использовать конструкции из спецификации `addrs.dbpl` в режиме «Только чтение». При этом, ссылаясь на описанные в `addrs.dbpl` таблицы и представления необходимо указывать псевдоним (ADRS.) в качестве префикса.

Второй способ интеграции (конструкция **INCLUDE**) позволяет объединить несколько спецификаций в одну. При этом может быть задан режим подключения: неявный, когда все предложения из указанной спецификации подключаются к главной, и явный, когда в определённые предложения добавляются неописанные в главной спецификации конструкции.

Выводы

В данной главе приведено описание этапов предлагаемой технологии разработки приложений БД. Важную роль при разработке играет модель ПБД, на основе которой автоматически создается (без компиляции) пользовательский интерфейс, обеспечивающий взаимодействие с предметной БД. Корректность объектов модели обеспечивается применением при создании специально разработанной инструментальной системы «ГеоАРМ». Для хранения и представления моделей ПБД разработан декларативный язык спецификаций – ЯПБД.

Выразительные средства разработанного языка обеспечивают возможность настройки большого перечня параметров приложения. С одной стороны, конструкции языка позволяют достаточно детально описать структуру БД (уровень Schema), с другой – позволяют представить эту информацию с точки зрения работы приложения, т.е. как информация из таблиц БД будет представлена пользователю (уровень Display). Кроме того, конструкции ЯПБД позволяют поддержать модульное программирование, т.е. при разработке спецификации приложений БД можно ссылаться на готовые спецификации других ПБД. Такая возможность позволяет создавать ПБД интегрирующие данные из нескольких предметных БД, а также ускорить процесс разработки в целом.

Лично автором разработаны технология создания ПБД, концептуальная модель ПБД и язык спецификаций ПБД (ЯПБД). Данные результаты опубликованы в работах [10, 11, 15, 47, 49, 53-55, 57, 58 60].

Глава 3. Инструментальная система создания приложений БД.

В третей главе определены функциональные и системные требования к инструментальной системе создания ПБД и представлена её программная архитектура.

3.1. Требования к функциональному обеспечению и архитектуре инструментальной системы

В рамках предложенной в главе 2 технологии создания ПБД на основе спецификаций сформулируем общие требования к функциональному, системному обеспечению и архитектуре инструментальной системы создания ПБД.

Предложенная в работе технология предполагает для создания ПБД четыре этапа, два из которых автоматизированы за счет применения инструментальной системы: создание модели ПБД в виде спецификации и настройка при помощи спецификации универсальной инструментальной системы «ГеоАРМ» на взаимодействие с предметной БД и внешними ППС, в том числе с ГИС. Предложенный в разделе 2.3 язык спецификаций является относительно простым и в принципе создание спецификаций ПБД может быть осуществлено через общедоступные текстовые редакторы. Однако для поддержания более высокого уровня автоматизации создания ПБД необходимы развитые инструментальные средства для создания спецификаций, которые позволяли бы использовать существующую в СУБД метаинформацию о предметной БД и расширять ее новой.

Каждое ПБД должно реализовывать выполнение четырех основных задач: создание, чтение, изменение, удаление данных. Важной задачей является и получение выборок данных по запросу. Для решения этих задач в приложении

должны быть реализованы визуальные пользовательские интерфейсы, настраиваемые по спецификации.

При взаимодействии пользователей с БД иногда возникает необходимость решения специфических задач, не заложенных в конфигурацию системы, например, построение сложных отчетов, решение вычислительных задач. Для решения таких задач в программном коде системы необходимо предусмотреть программный интерфейс, позволяющий создавать и затем подключать надстройки (внешние ППС), расширяющие функциональные возможности ПБД.

Часто в базах данных содержатся пространственные данные (например, почтовые адреса, географические координаты). В такой ситуации можно расширить функциональные возможности системы за счет интеграции с ГИС, что повысит наглядность представления данных, позволит решать пространственные задачи и, как следствие, повысит качество решений предметных специалистов. Для работы с пространственными данными необходим модуль, реализующий базовые ГИС-функции. При разработке такого модуля необходимо учесть, что цифровые карты часто содержат секретные данные или данные для служебного пользования.

Предложенная технология позволяет описать в спецификации структуру ряда подсистем ПБД. Для взаимодействия всех подсистем приложения между собой необходим модуль, реализующий внутреннее представление модели ПБД на основе интерпретации спецификации. Очевидно, этот же модуль должен обеспечивать трансляцию команд подсистем ПБД в команды драйверов для взаимодействия с различными СУБД.

Сформулируем функциональные требования к инструментальной системе создания ПБД:

- поддержка процесса создания и изменения спецификации ПБД;
- автоматическая настройка системы на работу с предметной БД по спецификации (интерпретация);
- обеспечение выполнения базовых задач работы с БД – создание, представление, изменение, удаление записей;

- поддержка создания пользовательских запросов;
- поддержка расширяемости возможностей ПБД за счет взаимодействия с надстройками (внешними ППС);
- поддержка взаимодействия с ГИС.

Системные требования, предъявляемые к инструментальной системе:

- обеспечение работы под ОС Windows;
- обеспечение работы с различными СУБД в технологиях ADO, BDE.

В результате сформулированных требований к функциональному обеспечение и архитектуре инструментальной системы создания приложений баз данных автором разработана инструментальная система «ГеоАРМ». В качестве среды разработки выбрана IDE Delphi. Выбор данной платформы объясняется наличием большого количества VCL-компонентов для разработки ПО для взаимодействия с БД и мощных средств отладки программного кода.

Архитектура инструментальной системы «ГеоАРМ» (Рисунок 7) включает следующие основные компоненты: подсистему управления спецификациями, ядро системы, подсистему редактор БД, построитель пользовательских запросов, программный интерфейс и подсистему «Карта» (как внешнюю ППС).

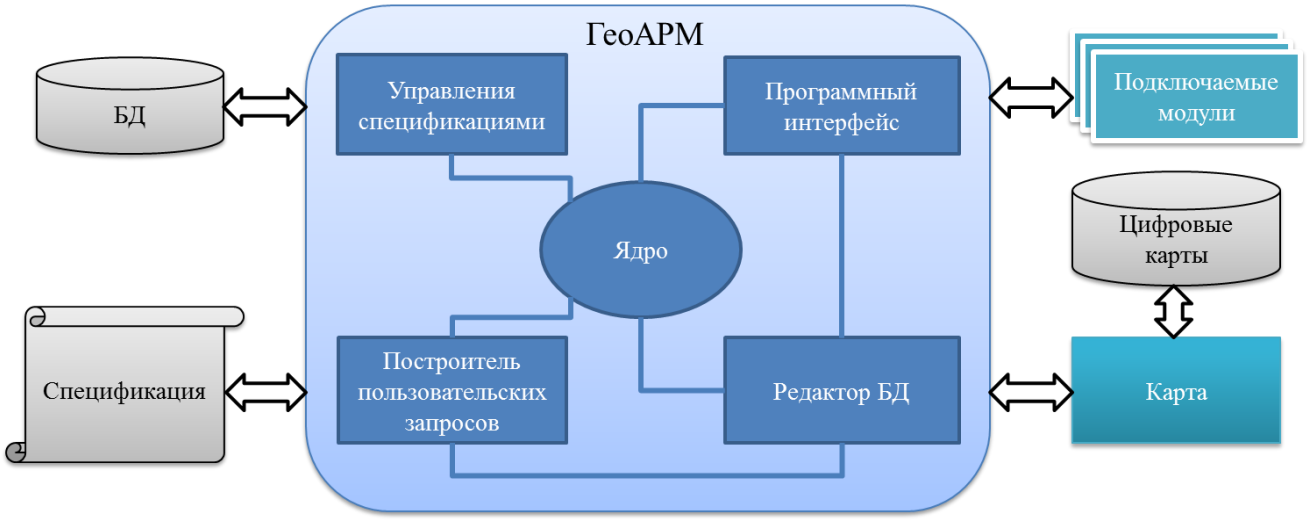

Рисунок 7. Архитектура инструментальной системы создания ПБД.

Общая схема разработки и исполнения ПБД в «ГеоАРМ» выглядит следующим образом. С помощью подсистемы управления спецификациями создаётся спецификация ПБД, спецификация загружается в ядро системы, в

результате чего инструментальная система становится предметным ПБД и обеспечивает возможность работы с таблицами БД через подсистемы «Редактор БД», «Построитель пользовательских запросов», взаимодействует с пространственными данными через подсистему «Карта» и позволяет решать специфические задачи при помощи внешних ППС подключаемых через встроенный программный интерфейс.

Рассмотрим детально структуру инструментальной системы «ГеоАРМ».

3.2. Ядро системы

Ядро системы отвечает за взаимодействии всех подсистем приложения с СУБД, обеспечивая интерпретацию спецификации ПБД и преобразование команд подсистем ПБД в команды интерфейсов для взаимодействия с СУБД. Для представления в памяти определённых в спецификации объектов модели ПБД в коде Ядра созданы соответствующие структуры данных.

Контекст приложения БД.

Класс TTblInfoCtx реализует представление общей информации о ПБД, а также обеспечивает создание соответствующих структур и компонентов для взаимодействия с БД. Свойства и флаги указанные в данной структуре распространяются на все объекты модели ПБД, если они не переопределены в структурах конкретных объектов.

Таблица 3. Основные поля класса TTblInfoCtx.

Поле	Тип	Назначение
AppName, AppTitle	String	Наименование и заголовок ПБД
Conn	TADOConnection	Соединение с БД
Schema	String	Схема используемой БД
SrcFiles	TStringList	Список файлов спецификаций ПБД.
Tables	TStringList	Список таблиц и представлений ПБД

GridEditMode	TGridEditMode = (geAuto,geForm,geTbl)	Режим редактирования записей по умолчанию - на форме, в таблице, или определять по количеству полей
luPKFmt, luNameFmt, QuoteFNFmt, QuoteQFNFmt, QuoteTNFmt:;	String	Формат формирования полей
QAlways, AllPKAutoInc, PreserveTblOrders, AutoIndex, PreviewDefaults, DBCaseIns, QFldAs, QJoinBraces, FilterMCB, BlobLookups, MFltAlways LogSQL, LogLU	boolean	Флаги
UpdateMode	TUpdateMode = (upWhereAll, upWhereChanged, upWhereKeyOnly)	Режим обновления
Dialect	TTblInfoSQLDialect = (sdUnknown, sdMSSQL, sdOracle)	Диалект SQL для формирования запросов к различным СУБД

MapLnkInfo	TObject	
MapLnkTables	TStringList	Список таблиц-связей с ЦК

Приведём описание основных методов.

function NewQuery(AOwner: TComponent): TQuery; — создать компонент запрос.

function GetTblInfoByName(const TN: String): TBaseTblInfo; — получить описание таблицы по имени.

function MapLnkTblIdByName(const MLTN: String): Integer; — получить номер таблицы связей из списка таблиц-связей с ЦК (MapLnkTables) по имени.

function GetTblInfoByMapKindEx(hMapTbl,Kind,hMinBase: integer): integer получить номер таблицы ПБД по номеру таблицы-связей с ЦК.

function AddSrcFileInfo(const FileName,ShortName: String): integer — добавить файл спецификации в список спецификаций SrcFiles.

function GetSourceName(hSrc: Integer): String; — получить имя файла спецификации по номеру из списка.

function GetSrcInfo(hSrc: Integer): TSrcFileInfo — получить файл спецификации по номеру.

function GetSourceSchema(hSrc: Integer): String — получить схему БД из файла спецификации.

function GetSrcTableByName(hSrc: Integer; Name: String): Integer — получить номер таблицы по номеру файла спецификации и имени таблицы.

function OpenDBInfoReader(AhSrc: Integer): TDBInfoReader — получить доступ к классу чтения спецификаций.

Представление информации о таблицах.

Для представление информации о таблицах БД используется иерархия классов, в которой базовым классом является тип TBaseTblInfo. Данный класс содержит общие свойства таблиц и представлений, а также абстрактные методы для получения и предоставления метаданных (списки полей, связей) таблиц БД. Наследниками данного класса являются классы TTblInfo и TViewInfo. TTblInfo

предназначен для взаимодействия с таблицами, а TViewInfo – представлениями (в терминах технологии).

Рассмотрим основные поля класса TBaseTblInfo (Таблица **4**).

Таблица 4. Поля класса TBaseTblInfo.

Поле	**Тип**	**Назначение**
Number	Integer	Номер таблицы
hSrc	Integer	Номер в списке файлов (TTblInfoCtx.SrcFiles), из которого прочитано описание таблицы
SemLinks	String	Массив связей ЦК
MapSemLnkInfo:	PMapSemLnkInfo	Описание связи полей таблицы с семантиками объектов ЦК
hMapTbl, MapKind, MapField	integer	Поля для описания связи с объектами ЦК через таблицу связей
Name, Descr	String	Имя таблицы (представления) ПБД. Русское имя таблицы
Filter	String	Строка фильтра.
FQuery	TQuery	Компонент запрос. Используется в режиме работы с БД через запросы.
hOrder	TFieldNumsNeg	Список полей сортировки.
hMainNames	TFieldNums	Именованные поля.
MainNamesSkipFlags	Integer	Флаг. Если именованное поле одно, то использовать для него автокомплит.
GridEditMode	TGridEditMode	Режим редактирования записей для конкретной таблицы. Перекрывает режим указанный в TTblInfoCtx

Тип TFieldNums = String используется для представления массивов полей, где код символа (ord(s[i])) соответствует номеру поля в списке полей таблицы (представления). TFieldNumsNeg = String позволяет представить массив полей со знаком, т.е. каждое поле шифруется парой символов: знак и номер поля.

Методы класса TBaseTblInfo условно можно разделить на 3 группы:

- методы, обеспечивающие взаимодействие с таблицами: формирование описания таблиц, установка атрибутов и режимов, создание наборов данных (TDataSet), включение фильтров и сортировок, формирование SQL-запросов;
- методы, обеспечивающие взаимодействие с полями таблиц: создание структур описания полей, получение атрибутов полей, установка режимов взаимодействия с полями, формирование и получение описания связей, формирование вида имён полей и задание значений при создании SQL-запросов;
- методы взаимодействия с объектами ЦК: связывание и получение данных об объектах ЦК.

Для каждой описанной в спецификации таблицы в памяти создаётся экземпляр TTblInfo = class(TBaseTblInfo), который представляет всю информацию о структуре и свойствах таблиц (реализует объект модели ПБД Table). На основе структурных данных из спецификации TTblInfo создаёт (метод CreateTTable) и связывает компонент TTable для работы с конкретной таблицей БД. Кроме того, TTblInfo позволяет получить информацию о всех связях текущей таблицы.

Экземпляры класса TViewInfo = class(TBaseTblInfo) создаются для описанных в спецификации представлений (в терминах технологии, глава 2) и соответственно хранят данные об их структуре. Так для создания представления в экземпляре TViewInfo хранятся данные о базовой таблице, на которой оно основано (BaseTbl: TTblInfo), а также реализованы методы декодировании связей для получения значений из полей подчиненных таблиц (представлений) и формирования связей в SQL-запросах.

Представление информации о полях.

Для представления информации о полях таблиц разработана следующая структура данных:

```
TFldInfo = object
    RName,                          Подпись поля
    sParms: String;                 Параметры
    Kind: TQFldKind;                Вид поля
    DisplW: Byte;                   Видимая ширена
    F: Integer;                     Флаги поля
end ;
```

В sParms хранятся указанные в спецификации свойства поля, которые влияют на отображение поля таблицы в графический интерфейс (например, цвет шрифта, размеры компонента).

Вид поля Kind – это значение типа TQFldKind = (qfNone, qfInt, qfFloat, qfName, qfString, qfPhone, qfRef, qfListed, qfBoolean, qfDate, qfImage, qfBLOB), которое определяется по виду поля в спецификации.

Включение конкретного флага (поле F) соответствует определённому биту в представлении целого числа (integer), что позволяет задать до 32 флагов. Определены следующие флаги (Таблица **5**):

Таблица 5. Флаги полей таблиц.

Флаг	**Назначение**
tfAutoinc = $1	Значение поля является счётчиком.
tfCast = $2	Тип поля не может быть неявно преобразован к указанному, поэтому во всех запросах необходимо использовать CAST.
tfReadOnly = $4	Поле только для чтения.
tfHasPlugInButton = $8	Рядом с полем должна быть кнопка вызова надстройки.
tfFileName = $10	Строковое поле, содержащее путь к файлу.
tfMultyline = $20	Поддержка многострочного режима для строковых полей (используется компонент TMemo вместо TDBEdit).

vfStarDef = $100	Поле представления было включено по ссылке на все поля (^*).
tfViewOnly = $400	Для блоб-полей, содержащих изображения защита от копирования.

Для полей представлений используется структура TVFldInfo расширяющая TFldInfo:

```
TVFldInfo = object(TFldInfo)
    QName: String;          Псевдоним поля.
    hFld: SmallInt;         Номер поля представления.
    hVLnk: ShortInt;        Номер ссылки.
end ;
```

Значение hVLnk содержит номер элемента массива связей TLinks = array[byte]of TLink, являющегося узлом в дереве связей для данной таблицы. Если поле является полем базовой таблицы данного представления, то hVLnk = -1.

Представление связей реализовано в следующей структуре:

```
TLink = record
    Name: String;           Имя ссылки.
    hSrcFld: TFieldNums;    Номера полей таблицы-источника.
    hTgtFld: TFieldNums;    Номера полей целевой таблицы.
    sCond: String;          Условие
    Tbl: TBaseTblInfo;      Целевая таблица.
end;
```

Данная структура позволяет представить связь между двумя таблицами как по одному, так и по нескольким полям. В hSrcFld хранятся номера полей таблицы, содержащих ссылки (являющихся FK). hTgtFld содержит номера полей целевой (справочника), а Tbl – описание самой целевой таблицы. В sCond может содержаться логическое условие на поля таблицы-источника, при котором ссылка имеет место.

Для описания ссылок представлений разработана структура TVLink.

TVLink = packed object

Tbl: TTblInfo;	Базовая таблица представления
hLnk: TLinkNum;	Номер ссылки
hParent: TLinkNum;	Номер
function GetTargetTbl: TBaseTblInfo;	Получить описание целевой таблицы.
function GetHSrcFld: TFieldNums;	Получить номер поля, содержащего ссылку.
function GetHTgtFld: TFieldNums;	Получить номер поля целевой таблицы.
function GetName: String;	Получить имя связи.

end;

Для представления связей типа «мастер-детали» разработана специальная структура TDetailLink:

TDetailLink = packed record

Name: String;	Наименование связи
Tbl: TBaseTblInfo;	Подчинённая таблица
hLnk: TLinkNums;	Путь ссылок.
F: Integer;	Флаги: dfHidden, dfChMaster

end

hLnk содержит путь ссылок из подчинённой таблицы в главную, что позволяет реализовывать взаимодействие с таблицами-деталями через несколько уровней ссылок.

Загрузчик ядра. Интерпретация спецификации.

Для формирования структур Ядра (описаний таблиц и представлений) разработана иерархия классов «загрузчиков» (Рисунок **8**).

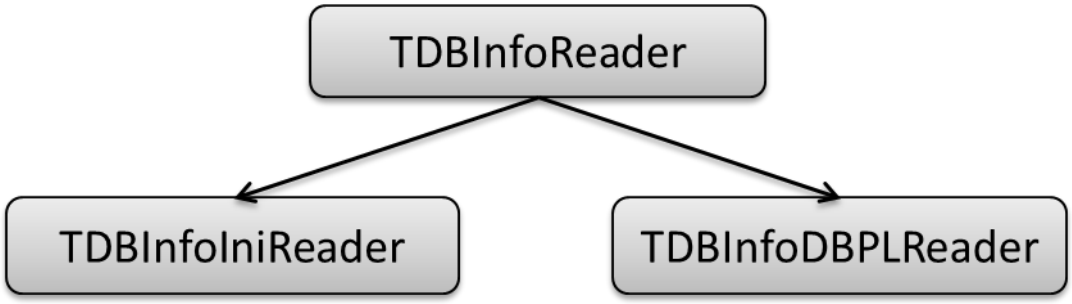

Рисунок 8. Иерархия классов загрузки

Базовый класс TDBInfoReader содержит описание абстрактных методов для интерпретации спецификаций ПБД, создания и загрузки соответствующих структур Ядра. Основные методы класса TDBInfoReader:

procedure DoLoadDBInfo; virtual; abstract; — создать контекст приложения.

procedure ReadUsedList; virtual; abstract; — получить связанные спецификации и добавить их в контекст приложения.

procedure ReadTblList; virtual; abstract; — создать список таблиц и представлений ПБД. (При этом создаются соответствующие пустые структуры TTblInfo, TViewInfo).

procedure LoadTblMenu(MainPar: Pointer; OnCreateItem: TCreateTblMenuItemEvent; IP: Pointer); virtual; abstract; — загрузить структуру меню сущностей ПБД.

procedure LoadTblInfo(TI: TTblInfo; var State: TTblInfoLoadState); virtual; abstract; — загрузить описание таблицы.

procedure LoadViewInfo(VI: TViewInfo; var State: TViewInfoLoadState); virtual; abstract; — загрузить описание представления.

procedure LoadTblFormInfo(TI: TBaseTblInfo); virtual; abstract; — загрузить описание пользовательской формы.

В наследниках класса TDBInfoReader — TDBInfoIniReader и TDBInfoDBPLReader описаны методы интерпретации для конкретных видов спецификаций. TDBInfoIniReader реализует загрузку из спецификаций ПБД в формате Ini-файлов, а TDBInfoDBPLReader – спецификаций на ЯПБД. Такой подход позволяет создавать новые классы загрузчики и тем самым поддержать

разнообразные формы представления модели ПБД. Например, в дальнейшем планируется реализация интерпретатора моделей ПБД представленных на UML.

Для разбора конструкций спецификаций ПБД представленных в ini-файлах в TDBInfoIniReader используются функции класса TCustomIniFile (поставляемый с Delphi, модуль IniFile.pas). Загрузка Ядра из спецификаций на ЯПБД требует предварительного синтаксического анализа и построения дерева разбора. Разработанный класс TDBPLAnalyzer реализует синтаксический анализ файлов спецификаций ПБД на ЯПБД и построение дерева разбора TDBPLTree. Методы загрузчика TDBInfoDBPLReader ориентированы на чтение дерева разбора TDBPLTree.

3.3. Подсистема управления спецификациями

Подсистема управления спецификациями приложений БД представляет собой визуальный пользовательский интерфейс, состоящий из набора визуальных инструментов, позволяющих решать различные задачи создания и модернизации спецификаций приложений БД:

- Настройка соединения с БД;
- Выбор и загрузка метаинформации из СУБД;
- Управление описанием таблиц;
- Управление описанием представлений;
- Описание надстроек;

Интерфейс настройки соединения (Рисунок **9**) позволяет задать общие параметры работы системы с БД. При этом задаётся технология доступа к БД: ADO или BDE. При выборе технологии ADO с помощью штатного диалога формируется строка соединения. Для доступа к БД в технологии BDE необходимо указать псевдоним базы, предварительно настроенный при помощи утилиты BDEAdministrator , а также имя пользователя и пароль.

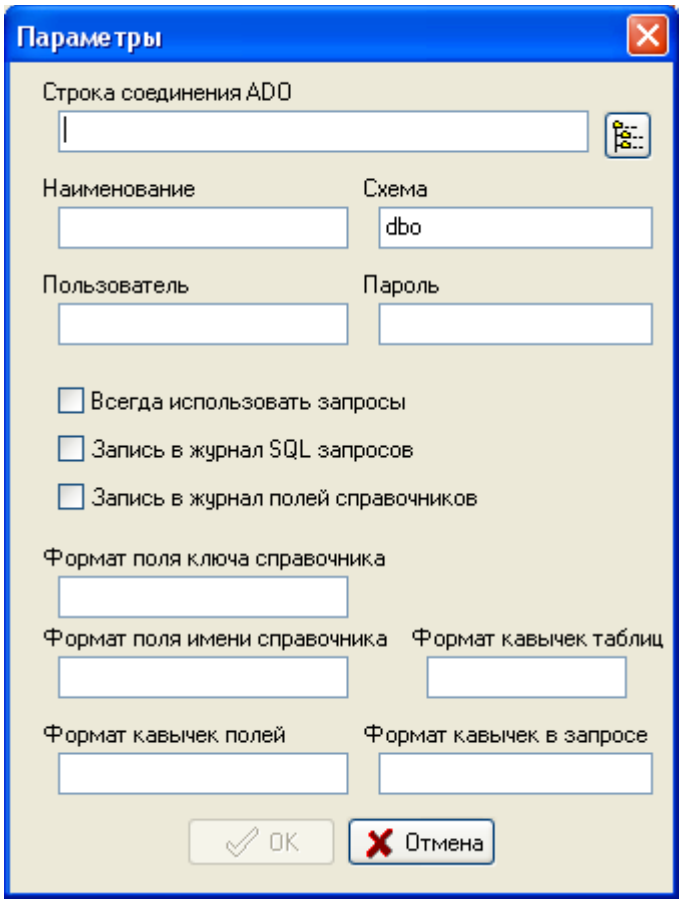

Рисунок 9. Параметры настройки соединения с БД

Кроме этого, при необходимости задаются некоторые общие параметры работы системы, такие как «Формат поля ключа справочника», «Формат поля имени справочника», «Формат кавычек полей», «Формат кавычек таблиц», «Формат кавычек в запросе». Все параметры настройки соединения записываются в спецификацию, после чего инструментальная система подключается к предметной БД.

После подключения к БД, предоставляемая СУБД метаинформация о таблицах, становится доступна инструментальной системе. При помощи модулей ПС Управления спецификациями (Рисунок **10**) администратор инструментальной системы может просматривать метаинформацию о структуре таблиц и включать её в спецификацию ПБД. Для удобства принятия решения о выборе таблицы в нижней части формы выводятся данные из текущей таблицы БД.

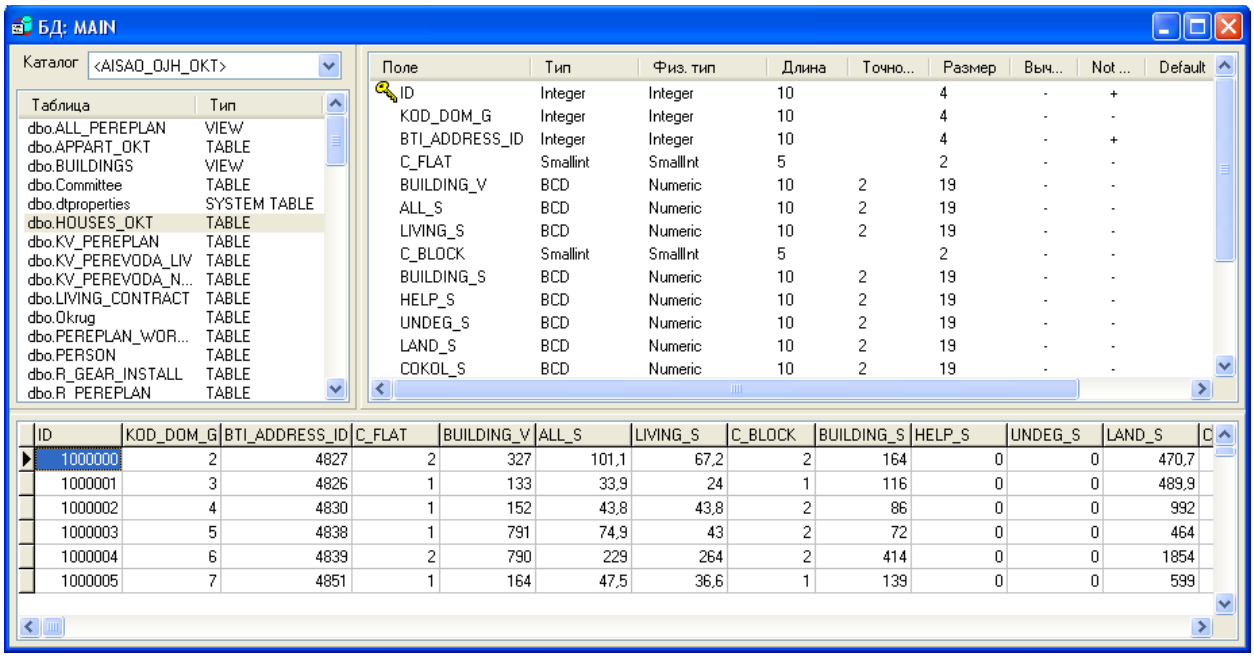

Рисунок 10. Интерфейс выбора и загрузки метаинформации о таблицах из СУБД.

Выбранные таблицы попадают в список таблиц спецификации приложения (Рисунок **11**), где метаинформация о них может быть расширена. Например, таблицам и полям таблиц указаны русские наименования, задан «Вид» полей (Рисунок **12**) (в терминах предлагаемой технологии см. главу 2), а также описана (расширена) информация о связях с другими таблицами, если такая информация отсутствовала в БД (Рисунок **13**).

Инструментальная система ГеоАРМ позволяет настраивать связи типа «один-ко-многим», причём связь необязательно должна быть с ключевым полем подчиненной таблицы. Также допускается описание связи из ключевого поля. При описании связи указывается наименование связи, выбираются из списков поле, в котором содержится ссылка, имя подчинённой таблицы или представления и поле подчинённой таблицы или представления, в которое указывает ссылка. При этом можно установить флаг о наличии связи типа «мастер-детали» у подчиненной таблицы.

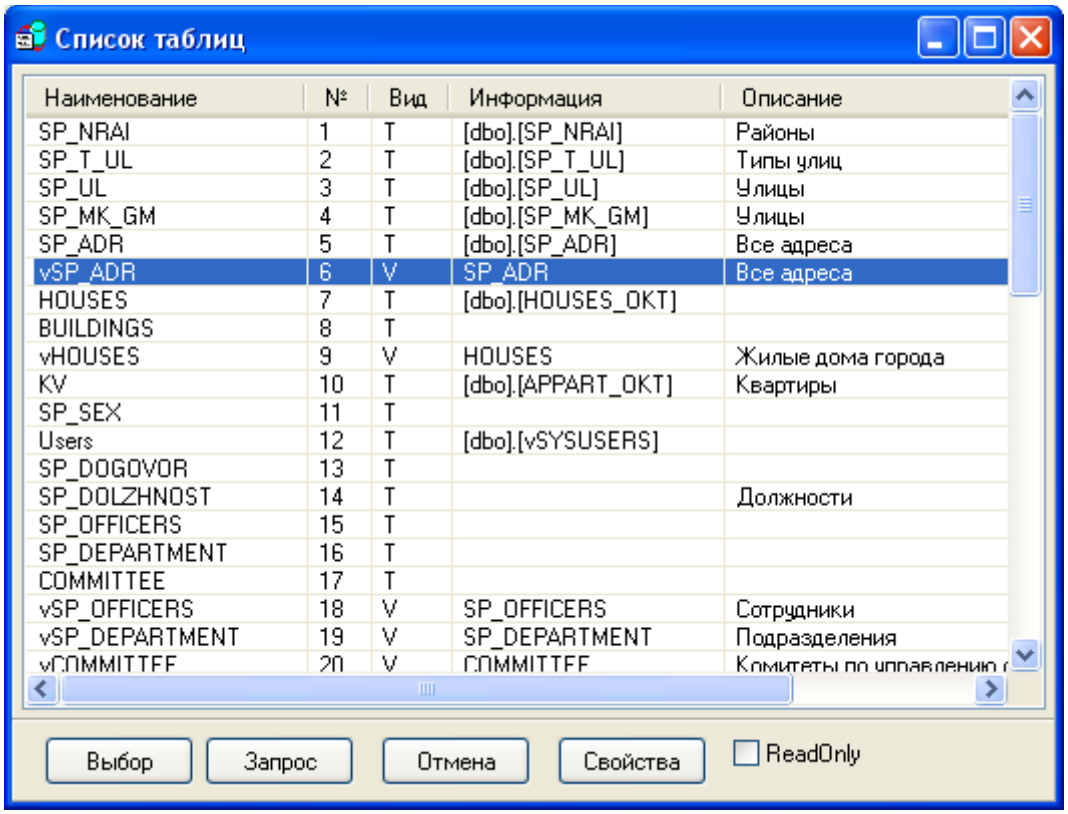

Рисунок 11. Список таблиц спецификации

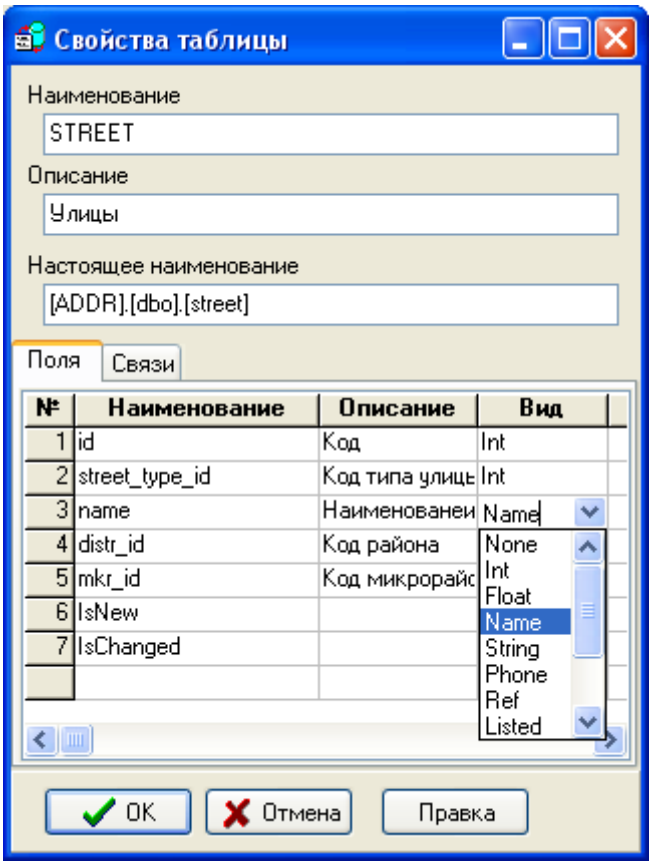

Рисунок 12. Окно настройки свойств полей таблицы

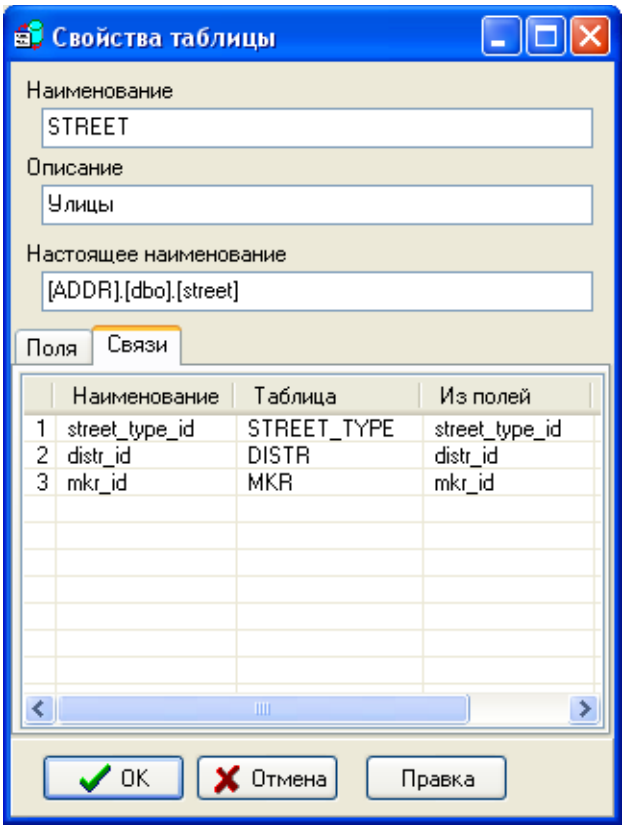

Рисунок 13. Окно настройки связей таблицы

При наличии информации о связях, на основе таблиц, при необходимости, задаются представления. В редакторе представлений (Рисунок **14**) поля из базовой и подчинённых таблиц представлены в виде дерева разбора. Администратор может выбрать, какие поля из дерева будут содержаться в представлении.

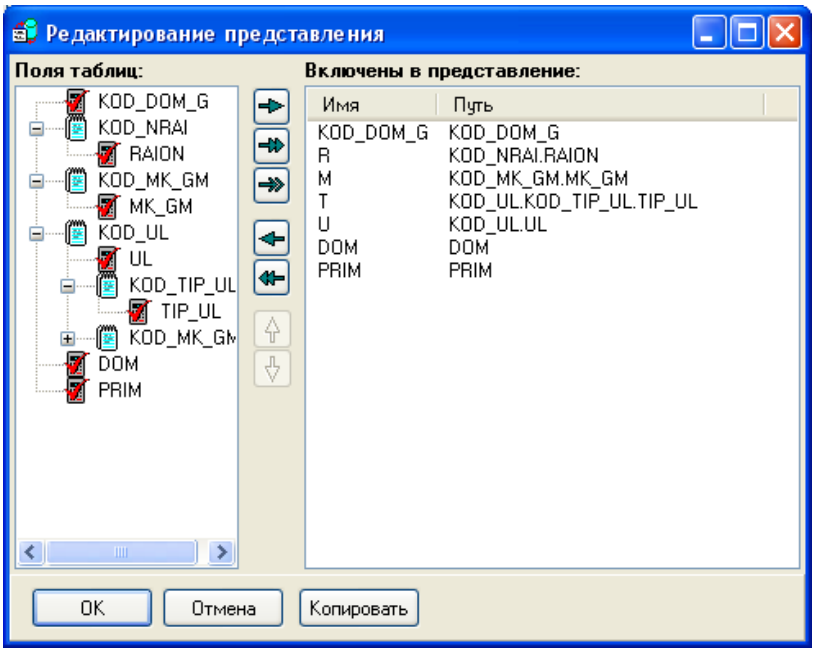

Рисунок 14. Редактор представлений

3.4. Подсистема Редактор БД

Подсистема Редактор БД представляет собой динамический интерфейс для работы с БД. Редактор состоит из трех частей: области меню сущностей, области задач и области данных. В области меню сущностей (рисунок 15) отображаются описанные в спецификации имена таблиц и представлений приложения БД, объединенные в смысловые группы, исходя из их назначения или функций.

В области задач отображаются элементы управления для решения различных задач обработки данных. Состав элементов управления для каждой таблицы или представления регулируется спецификацией. Для всех таблиц и представлений всегда присутствуют кнопки навигации (в режиме только на чтение кнопки модификации не активны), кнопка «Правка» для перехода в режим просмотра записи в виде формы, кнопка «Экспорт» и кнопка «Запрос». Опционально могут присутствовать кнопки вызова надстроек и взаимодействия с электронной картой. Расстановкой элементов управления в области задач управляет менеджер компоновки.

Область данных служит для отображения информации из выбранных в области меню таблиц или представлений. Информация из таблиц и представлений отображается в двух режимах: в табличном и в виде формы.

В табличном режиме пользователь может просматривать и редактировать информацию из таблиц и представлений. Причем, для представлений поддержан выбор значений из списка для именованных полей, полученных по ссылкам. Например, поле «Улица» в представлении «Адреса» (рисунок 15) можно выбрать из списка, так как это именованное поле представления «Улицы», полученное по ссылке. При этом поле «Тип» изменится автоматически в зависимости от того, какому типу принадлежит выбранная улица.

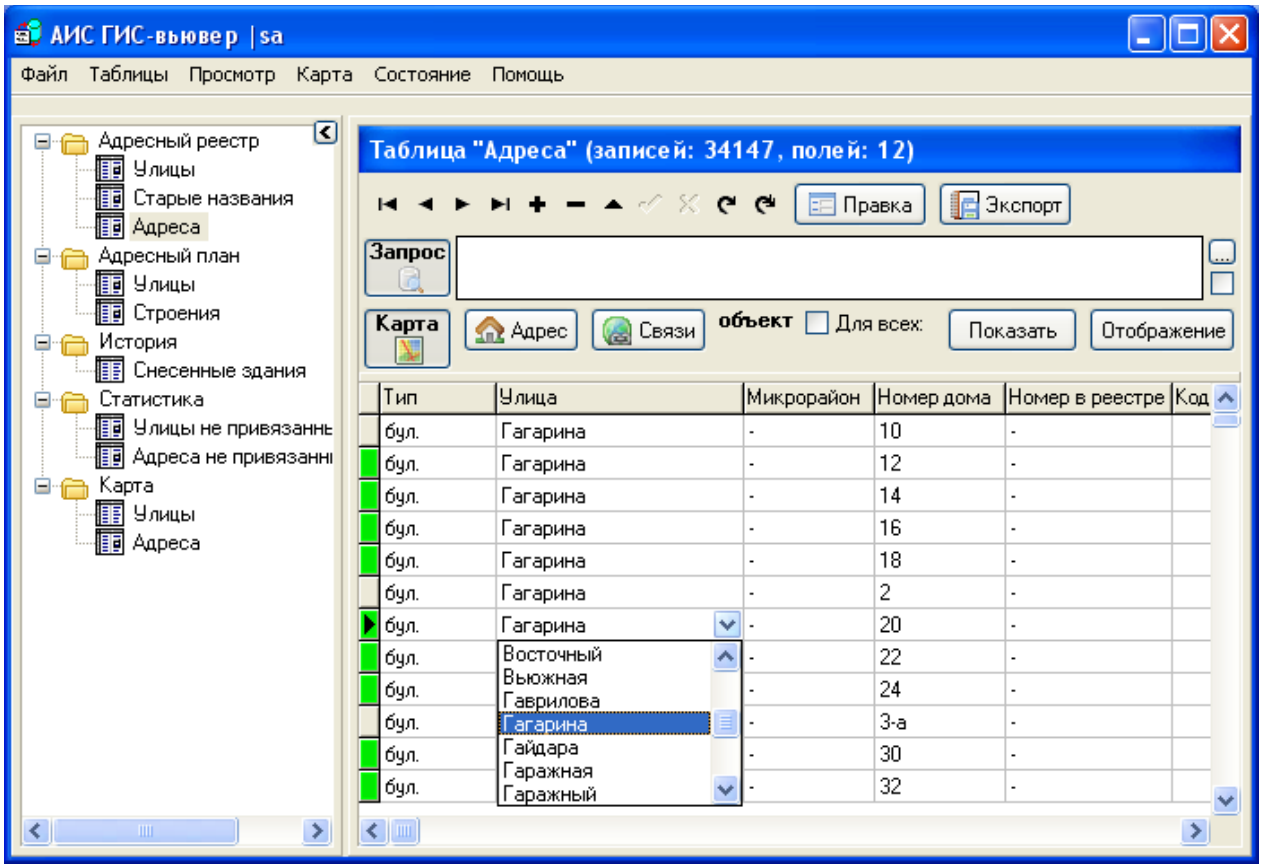

Рисунок 15. Редактор БД

Порядок и ширину столбцов может задать непосредственно сам пользователь исходя из своих предпочтений. При этом настройки для каждого пользователя сохраняются в виде wsf-файла [89] в его профиле.

Режим формы – это универсальный интерфейс для работы с записями через набор, связанных с атрибутами таблиц или представлений, визуальных компонентов. Режим формы активизируется двойным нажатием левой кнопки мыши по записи, нажатием кнопки «Правка» или автоматически при создании новой записи (если количество атрибутов достаточно большое). На форме каждому полю записи таблицы или представления на основе информации из спецификации сопоставляется набор визуальных компонентов. Набор и типы компонентов определяются в зависимости от вида, атрибутов поля и того, как оно получено, из базовой таблицы или по ссылке. Каждому полю всегда сопоставляются компоненты – идентификатор и модификатор поля. Компонент идентификатор (TLable) отображает информацию о русскоязычном имени поля таблицы, а также может содержать информацию о принадлежности поля к

первичному ключу и требования обязательного наличия значения в поле. Компоненты модификаторы – это визуальные компоненты, с помощью которых значение в поле таблицы или представления может быть изменено.

За автоматическую компоновку элементов на форме отвечает менеджер компоновки, управляющий размерами и положением визуальных компонентов при изменении размеров формы. Под автоматической компоновкой элементов управления понимается автоматическое формирование пользовательского интерфейса, т.е. расстановка элементов управления на форме. При этом элементы управления должны располагаться таким образом, чтобы пользователю было удобно с ними взаимодействовать. Например, поля для ввода и отображения информации должны максимально полно отображать данные из соответствующих полей таблиц БД, но при этом желательно, чтобы они все находились в поле зрения пользователя. При изменении размеров пользовательской формы или разрешения экрана компоненты должны перераспределяться заново наиболее удобным для восприятия пользователя образом.

Менеджер размещения компонентов на форме.

При разработке менеджеров компоновки, за основу была взята идея менеджера компоновки FlowLayout Java [76], т.е. менеджера, который укладывает компоненты в ряды друг за другом, слева на право. Если в FlowLayout размер у каждого компонента фиксированный, то в разработанном менеджере было положено, что каждый объект имеет три возможных размера: минимальный (szMin), лучший (szBest) и максимальный (szMax). Максимальный размер — это размер компонента, соответствующий размеру поля таблицы БД. Например, полю типа varchar(150) будет соответствовать максимальный размер компонента, отображающий 150 символов. Было замечено, что часто для хорошего восприятия информации достаточно отображать часть поля таблицы, так как информация в них занимает не все поле целиком. Поэтому для каждого объекта можно вычислить лучший размер (70% от размера поля). Минимальный размер компонента – это размер, при котором читаема хотя бы часть данных (8-10

символов). Кроме того, разработанный менеджер стремится поджать ряды компонентов к верху окна.

В спецификации приложения БД содержится информация о способах отображения данных. Так, в описании отображения полей некоторого представления могут быть указаны как отдельные поля, так группы полей, которые на форме отображаются в виде набора полей, объединенных в контейнер (например, GroupBox).

По своей сути, пользовательская форма – это контейнер (главный контейнер) для элементов управления, на котором могут располагаться как отдельные элементы, так другие контейнеры (вложенные контейнеры) и т.д. Разработанный менеджер компоновки (рисунок 16) автоматически размещает элементы управления в контейнере по следующему алгоритму. Сначала менеджер компоновки получает размеры (szMin, szBest, szMax) всех элементов контейнера и пытается разместить их с максимальными размерами друг за другом, слева на право, сверху вниз. Если при таком размещении все компоненты и контейнеры остаются видимыми на форме, то компоновка останавливается. Иначе, менеджер компоновки пытается разместить компоненты с лучшими (szBest) размерами, при этом, если в какой-то из строк остается место, недостаточное для следующего компонента, то менеджер компоновки проверяет, хватит ли этого места для размещения одного или нескольких компонент данного ряда с максимальными (szMax) размерами. При этом приоритет имеют компоненты с наибольшими максимальными размерами. Если компоновка с лучшими размерами не дает необходимого результата, менеджер расставляет компоненты с минимальными размерами.

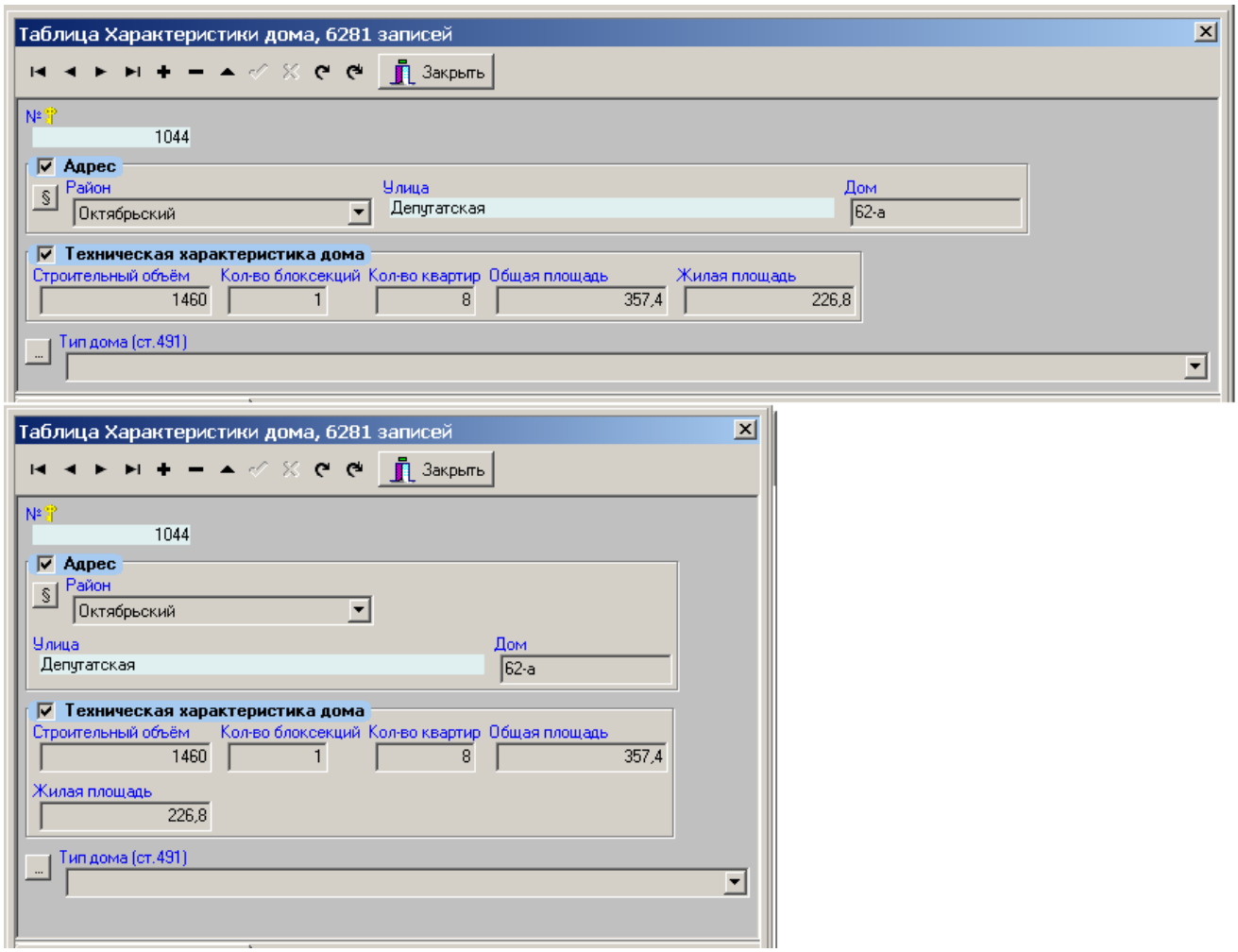

Рисунок 16. Пример работы разработанного менеджера компоновки при изменении размеров формы

3.5. Построитель пользовательских запросов

При разработке большинства ИС, предоставляющих пользователю доступ в БД, одной из важных задач является реализация механизмов поиска информации. Эффективный механизм поиска не только обеспечивает переход к интересующим пользователя записям, но и позволяет предметному специалисту самостоятельно получать выборки информации из БД, не прибегая к помощи специалистов по информационным технологиям.

В данной работе предложен механизм построения запросов [62], который, с одной стороны, использует интерактивное редактирование условий на значения полей, подобное тому, которое реализуется в поисковых формах, а, с другой

стороны, позволяет задавать достаточно сложные условия, конечно, не настолько произвольные как при непосредственном редактировании текста запроса, но все же достаточно разнообразные для решения большинства возникающих перед пользователем задач. Построитель запросов является универсальной формой, которая настраивается на работу с конкретной таблицей при помощи спецификаций приложения БД.

Для того чтобы при формировании условия запроса пользователь мог свободно комбинировать ограничения на значения отдельных полей, необходимо поддержать редактирование произвольных логических выражений. Внутренним представлением выражения является его дерево разбора, пример которого показан на рисунке 17. Для интерактивного редактирования выражения его можно представить в виде подобного дерева. Однако если в узлах дерева потребуется отобразить более длинные надписи, вертикальное расположение дерева окажется неприемлемым: надписи начнут накладываться друг на друга, или потребуется рассредоточить узлы, что затруднит понимание логики, задаваемой деревом разбора выражения. Более компактное размещение надписей можно получить, если направить их по вертикали. Но, поскольку горизонтально расположенный текст воспринимается лучше, чем вертикальный, того же эффекта лучше достичь за счет горизонтального размещения всего дерева.

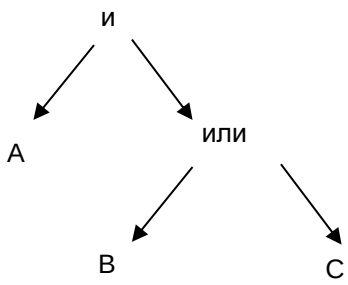

Рис. 17. Дерево разбора выражения A и (B или C)

Для редактирования выражений в виде горизонтально ориентированного дерева был разработан элемент управления TExprEditor (рисунок 18). Компонент поддерживает три способа редактирования значения в узле дерева, представляющего выражение: редактирование на месте; редактирование в отдельном диалоге и редактирование на той же форме. Помимо редактирования содержимого узлов необходимо иметь возможность изменения их взаимного расположения. Для этих целей в компоненте поддерживаются операции

изменения уровня вложенности операторов (соответствующие перестановке скобок в выражении), операция перестановки операндов и ряд других.

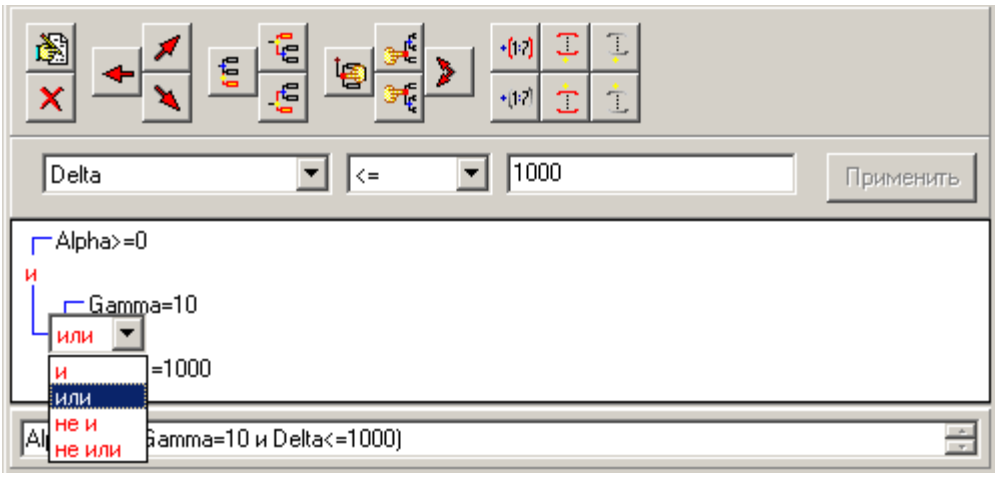

Рисунок 18. Компонент TExprEditor

Для редактирования условий запросов реализован построитель запросов, который может работать как в упрощённом, так и в расширенном режиме. В упрощённом режиме пользователь видит таблицу с именами полей, в которой он может задать ограничения на значения некоторых из них, при этом конъюнкция условий образует условие запроса.

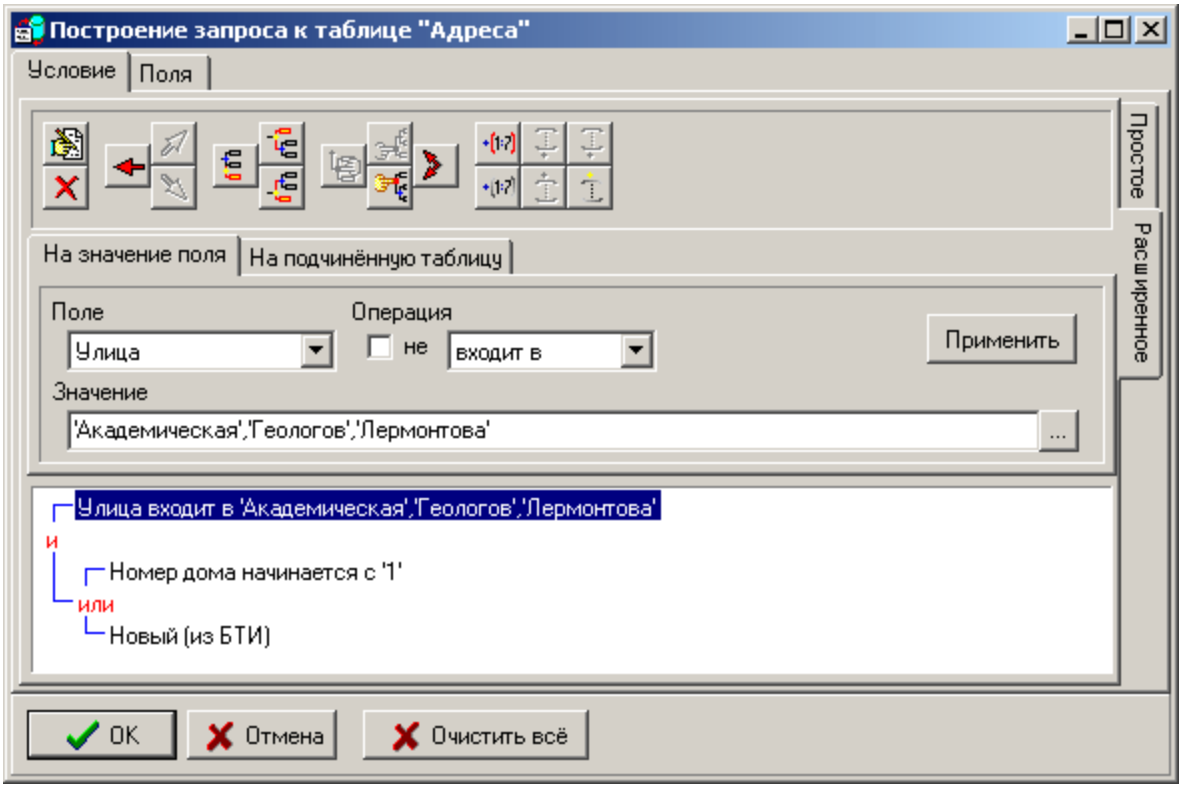

Рисунок 19. Расширенный режим построителя запросов

При необходимости задать более сложное условие можно перейти в расширенный режим (Рисунок **19**), в котором используется компонент TExprEditor. В любом режиме способ редактирования условий на значения поля определяется информацией о типе поля из спецификации структуры БД. Например, для списочных и именованных полей будет доступен режим выбора значений из списка.

В расширенном режиме поддерживается формирование условий на записи подчинённых таблиц (Рисунок **20**). При формировании таких условий построитель запросов вызывается рекурсивно. В приведённом на рисунке примере формируется условие: «Район города входит в 'Ленинский р-н', 'Свердловский р-н' и Количество записей в таблице "Адреса" с (В таблице "Здания по адресу" есть записи с (Тип здания входит в 'АЗС')) >=2». Иными словами: «На каких улицах Ленинского и Свердловского районов находится более двух АЗС?». При этом используется два уровня подчинённости таблиц: Улицы содержат Адреса, а по одному Адресу может находиться несколько Зданий. Для задания условий на подчинённые таблицы формируются коррелированные подзапросы, в часть WHERE которых включаются условия связи мастер-детали. В рассматриваемом примере формируется следующий текст на SQL:

```
select T.id Id,T.reg_id reg_id,T0.type ST,T.name
name,T1.name Distr,T2.name mk,T.BTI_id BTI_id,T.IsNew
IsNew,T.IsChanged IsChanged from dbo.street T
  LEFT OUTER JOIN dbo.street_type T0 ON
(T0.Id=T.street_type_id)
  LEFT OUTER JOIN dbo.distr T1 ON (T1.id=T.distr_id)
  LEFT OUTER JOIN dbo.mkr T2 ON (T2.id=T.mkr_id)
  WHERE (T1.id IN (8,10) and (select count(*) from
dbo.buildings S
WHERE (T.id=S.street_id)and(exists(select * from dbo.bld R
```

```
WHERE      (S.Id=R.ADDRESS_ID)and(UPPER(R.HOUSE_TYPE)      IN
('АЗС'))))>=2)
order by T.id
```

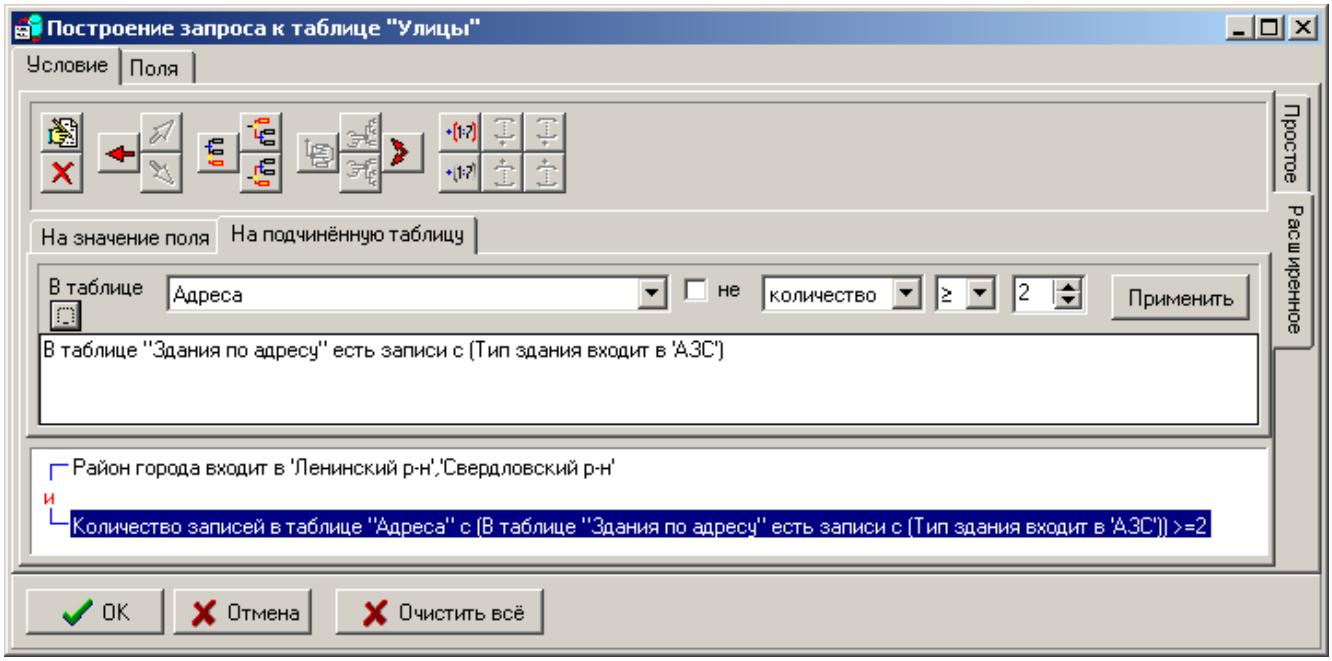

Рисунок 20. Логическое условие на количество записей подчинённой таблицы, удовлетворяющих требованиям

Написание подобного текста вручную займёт немало времени даже у опытного программиста. Несмотря на то, что формируемые запросы ориентированы на выбор записей одной (базовой) таблицы, в их условиях может использоваться информация из связанных с данной таблицей справочников и подчинённых таблиц, в результате чего предложенный механизм обладает большой выразительной силой.

3.6. Программный интерфейс для взаимодействия с внешними программными системами

Для реализации специфических функций, не заложенных в конфигурацию (например, построения сложных отчетов, решения вычислительных задач, конвертации данных в определенные форматы), в инструментальной системе

реализован программный интерфейс, позволяющий взаимодействовать с внешними программными системами (приложениями-надстройками).

В спецификации ПБД задается, для каких таблиц (представлений) и с какими параметрами вызываются внешние ППС. Программный интерфейс приложениям-надстройкам предоставляет ряд функций для доступа и взаимодействия с данными определенных таблиц или представлений. В таблице 6 представлены функции интерфейса и их назначение.

Таблица 6. Функции интерфейса

Функции	**Описание**
TGetConnHandleProc	получить информацию о соединении с БД
TGetTableNumByNameProc	получить набор данных
TGetTableCursorProc	получить курсор набора данных
TGetTableFieldCountProc	узнать количество полей
TGetFieldNumByNameProc	получить номер поля по имени поля
TGetFieldNameByNumProc	получить имя поля по номеру поля
TGetFieldValProc	получить значение из поля по номеру поля
TSetFieldStrValProc	получить значение из строкового поля по номеру поля
TSaveBlobToFileProc	сохранить информацию из BLOB-поля в файл; TGetBlobStreamProc – передать в потоке данные из BLOB-поля
TOpenTableIteratorProc	получить итератор по выбранным в таблице записям
TOpenDetailIteratorProc	получить доступ к итератору по подчинённым таблицам. Используется для перебора записей из таблиц деталей
TOpenSubDetailIteratorProc	получить итератор по подчинённым

	таблицам подчинённых таблиц
TDetailIteratorNextProc	получить следующую запись (шаг итератора)
TDetailIteratorGetFieldNumByNameProc	получить номер поля по имени поля для записи из таблицы деталей
TDetailIteratorGetFieldValProc	получить значение из поля по номеру поля для записи из таблицы деталей
TCloseDetailIteratorProc	закрыть итератор

В программной архитектуре приложения-надстройки должна быть реализована процедура инициализации (Init), в которой передается указатель на ПБД и происходит связывание процедур интерфейса с процедурами реализации выполнения, передачи строковых значений и завершения программы приложения-надстройки. В процедуре реализуется решение задачи, а в процедуре завершения происходит очистка памяти. В процедуре передачи строковых значений должно быть реализовано преобразование указателя на строку к определенному виду. При вызове в ПБД внешней программной системы сначала происходит инициализация, а далее запускается выполнение задачи. В процессе выполнения задачи приложение-надстройка, вызывая функции программного интерфейса (таблица 6), получает доступ к необходимым таблицам и представлениям, информацию из определенных атрибутов и может осуществлять перебор записей из таблиц-деталей (взаимодействие с таблицами-деталями ограничено глубиной в два уровня). В общем виде схема решения задач в приложениях надстройках выглядит следующим образом (рисунок 21).

Рисунок 21. Схема решения задач приложения-надстройки

Представленный набор методов позволяет организовать взаимодействие ПБД с внешними ППС для решения большинства задач обработки, представления данных или вычислений. В частности, данный механизм был применён для построения пользовательских отчётов для таблиц и представлений. Пример приложения-надстройки, реализующей вычислительную задачу, приведён в 4.2.

3.7.Построитель отчётов

Генератор отчётов разработанный в рамках данной работы реализован в виде надстройки (библиотека DLL) и получает данные через программный

интерфейс, что позволяет расширять функции построителя без изменения кода всей системы. Учитывая широкую популярность MS Word, в качестве шаблонов построитель отчётов использует шаблоны в MS Word, содержащие элементы разметки текста (метки).

При вызове генератора из спецификации ПБД передается информация о соединении с БД, файле-шаблоне и таблице или представлении, для записей которых нужно построить отчет. В процессе подготовки отчета происходит нахождение меток в тексте шаблона, после чего на основании параметров метки происходит вставка данных.

Метки шаблонов отчётов имеют следующий вид:

<Метка>:="<#"<Вид данных> <Место> [<Формат>] ">"

<Вид данных> := "FV"| "FIO" | "IMG" | "DETAIL"

<Место> := <Имя поля> | <ФИО> | <Имя деталей>

<ФИО> := "SN="<Имя поля> "FN="<Имя поля> "MN="<Имя поля>

<Имя поля> := "N="<Буква>*|<Цифра>* – имя поля таблицы, из которого вставляются в отчёт данные;

<Имя деталей> := "N="<Буква>*|<Цифра>*

<Формат> := <Падеж> | "SHOW"

<Падеж> := "PDG=" ("IP"|"RP"|"DP"|"VP"|"DP"|"PP")

FV – значение из поля таблицы;

FIO – значения из полей Фамилия, Имя, Отчество (далее должны следовать метки для получения имён полей содержащих ФИО);

IMG – изображение из Blob-поля таблицы

DETAIL – указывает, что в этом месте будут вставлены значения полей таблицы деталей.

Для реализации склонения по падежам применена библиотека padeg.dll [41].

Кроме того в отчёте реализован условный вывод данных. Для этого применяются следующие метки

```
"<# IF" <Имя поля> <Значение> ">"  <Вывод> "<#ELSE>"
                    <Вывод> "<#END>"
<Значение> := "V="<Буква>*|<Цифра>*
<Вывод> := <Метка> | (<Буква>*|<Цифра>*)
```

3.8. Подсистема «Карта»

Подсистема «Карта» реализована при помощи разработанного автором инструментального средства для обеспечения взаимодействия с пространственными данными – «Интерфейса АП» [16, 48, 50]. Данное средство разработано с использованием пакета GIS ToolKit [42] ГИС «Панорама» и представляет собой COM-сервер, вызов методов которого позволяет создать окно для взаимодействия с цифровыми картами (ЦК) и обеспечивает выполнение ряда функций обработки ПД. Использование «Интерфейса АП» при разработке или модернизации существующих ППС позволяет избежать реализации функций работы с цифровой картой. Кроме того, данное приложение может использоваться независимо от каких-либо ППС, как средство работы с цифровыми картами.

Важной особенностью реализации модуля для работы с ПД в виде COM-сервера является возможность настройки средствами ОС Windows запуска от имени пользователя, обладающего правами доступа к файлам карты, что позволяет исключить несанкционированное копирование файла электронной карты. В режиме «по умолчанию» COM-приложения запускаются под учетной записью текущего (запускающего) пользователя, но есть возможность при помощи утилиты dcomcnfg.exe ОС Windows настроить запуск под учетной записью указанного пользователя. Таким образом, данная возможность позволяет наладить работу с файлами карт, доступ к которым текущему пользователю запрещен.

«Интерфейс АП» обеспечивает выполнение стандартных функций ГИС, таких как масштабирование, перетаскивание, получение информации по объекту,

создание пользовательских слоев, вывод на печать. Кроме того, в данной системе реализована функция «Адресный план» – поиск объектов на цифровой карте по адресу.

Таблица типов «Интерфейса АП» содержит описание двух программных интерфейсов IApplication и IClient. IApplication содержит описание функций для работы с ПД, реализованные в самой системе, а IClient – функции, обеспечивающие передачу значений из «Интерфейса АП» в программу-клиент, которые должны быть реализованы на клиенте, соответственно. Функции интерфейсов приведены в таблице 7 и таблице 8.

Таблица 7 Функции интерфейса IApplication

Функции	**Описание**
function ShowObjByAddr(const S: WideString; const H: WideString): HResult;	Поиск объектов на ЦК по атрибутам «Улица, Дом»
function ShowObjByID(ID: Integer): HResult;	Поиск объекта на ЦК по уникальному номеру
function ShowObjsByAddrs(const Addrs: WideString; L: Integer): HResult;	Поиск списка объектов на ЦК по атрибутам «Улица, Дом». В Параметр Addrs передаётся список адресов типа «Улица1, Дом1, …,УлицаN, ДомN», а параметр L=N.
function ShowObjsBySemNum(SN: Integer; const SV: WideString): HResult;	Поиск объектов на ЦК по значению их семантики. Здесь SN номер семантики в классификаторе ЦК, а SV – значение.
function AddObjsToList(IDL: Integer; IDOBJ: Integer; SelAppend: Integer; Color: Integer): HResult;	Добавить объект ЦК в список выделения для последующего отображения. Здесь IDL номер слоя ЦК, IDOBJ – уникальный номер объекта

function ShowSelList: HResult;	Отобразить список выделения.
function ResetSelList(Color: Integer): HResult;	Очистить список выделения.
function SetClientInt(const ClientInt: IClient; SN: Integer): HResult;	Передать приложению-серверу указатель на интерфейс клиента.
function AddClientButton(Tg: Integer; const Hnt: WideString; SN: Integer): HResult;	Создать в интерфейсе пользователя приложения-сервера кнопку. Здесь Tg – номер кнопки, Hnt – всплывающая подсказка кнопки, SN – номер семантики.
function SetButtonBitmapData(Tg: Integer; Pic: OleVariant): HResult;	Задать изображение для кнопки в потоке. Здесь Tg – номер кнопки, которой задаётся изображение, Pic – изображение.
function SetButtonBitmapFromFile (Tg: Integer; const FileName: WideString): HResult;	Задать изображение для кнопки из файла.
function SetButtonBitmapFromRes (Tg: Integer; const ExeName: WideString; const ResName: WideString): HResult;	Задать изображение для кнопки из ресурса.
function GetSelObjID(var L: HResult): HResult;	Получить уникальный номер и номер слоя выделенного объекта.

Таблица 8 Функции интерфейса IClient

Функции	**Описание**
procedure GetObjInfoEvent(const S: WideString; const D: WideString; const V: WideString);	Получение информации по объекту ЦК
procedure GetObjInfoPress(Tg:	Обработка нажатия клиентских кнопок

`Integer; const VL: WideString);`	«Интерфейса АП».
function `GetFieldValByNameClnt (const FldName: WideString)`	Получить значение поля по имени.
procedure `ShowMapObjDBInfo(hL: Integer; hObj: Integer)`	Показать в приложении-клиенте информацию связанную с объектом ЦК.
function `DisconnSrv: HResult;`	Обработка закрытия окна «Интерфейса АП»
function `ObjLinkDelete(ObjID: Integer; LID: Integer): HResult;`	Обработка удаления объекта с ЦК. Необходимо для удаления связей на клиенте с объектами ЦК.

Привязка записей БД к объектам ЦК осуществляется двумя способами: через геокодирование, т.е. при помощи полей содержащих адреса домов, или привязкой к произвольным объектам карты через таблицу связей. Для первого способа спецификации таблицы (представления) БД указываются поля таблиц, в которых содержатся значения адреса объекта (например, «ADDRS=UL,Dom») При наличии такой информации на форме для редактирования таблицы отображается кнопка «Адрес», после нажатия на которую выполняется поиск адреса. При этом открывается окно карты, улица и номер дома заносятся в поля панели поиска адреса, и, если такой адрес имеется на карте, то в окне выделяется соответствующее здание (Рисунок **22**). Для использования второго способа в БД должна быть создана специальная таблица привязки объектов к карте MAP_LNK (одна на всю базу), позволяющая связать любой объект карты с данными из БД (Таблица 9).

Таблица 9 Структура таблицы привязки MAP_LNK

KIND	Код таблицы в БД
ID	Указатель на ключевое поле таблицы с данными из тематической БД (таблица определяется значением поля KIND)
ID_L	Номер слоя карты
ID_OBJ	Уникальный номер объекта на карте в слое ID_L

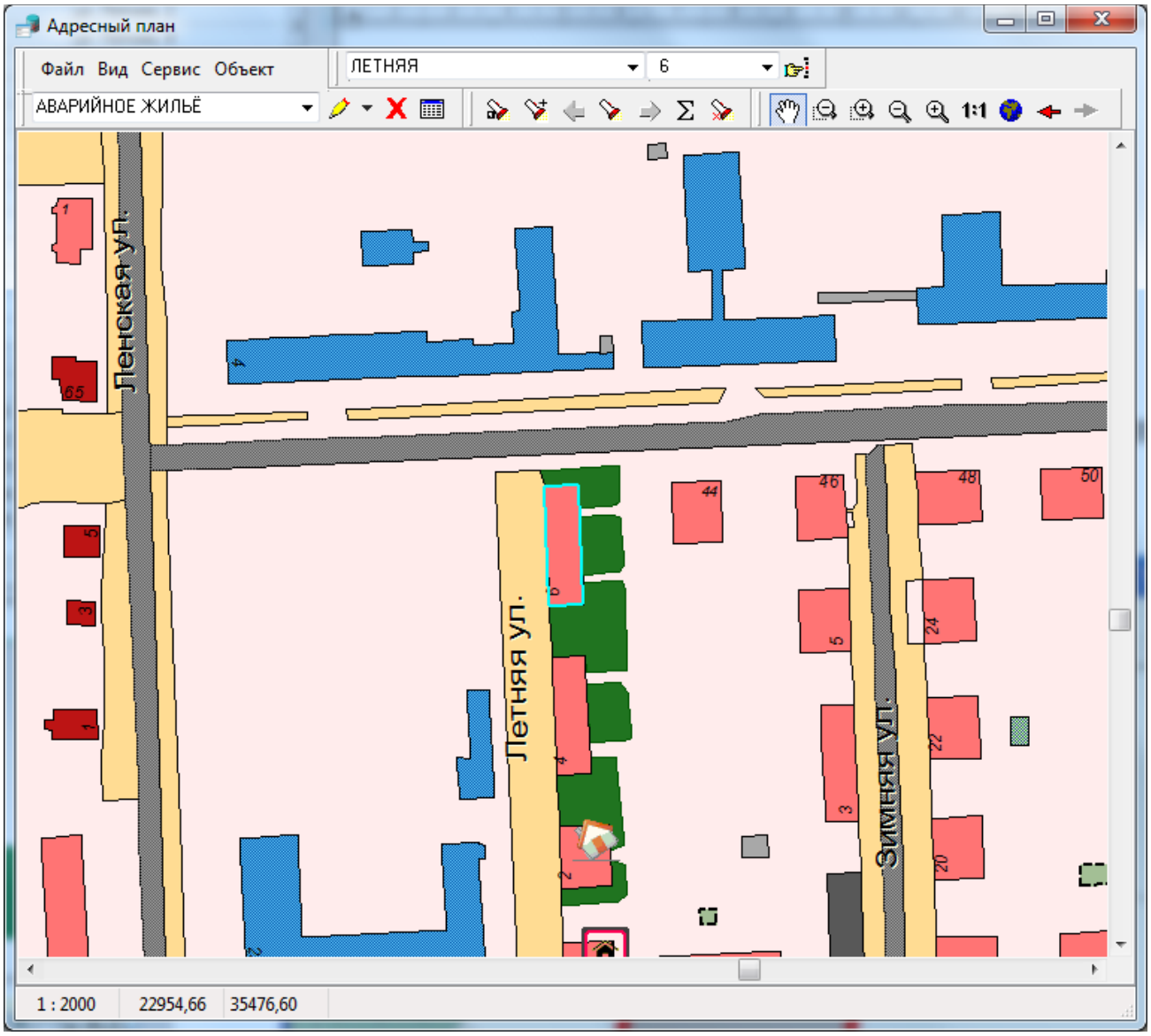

Рисунок 22. Окно подсистемы «Карта»

При работе с модулем «Карта» поверх основной топоосновы могут создаваться специализированные (пользовательские) слои, предназначенные для нанесения на карту отсутствующих там объектов (например, слои «Гаражи», «Реклама» и т.д.). При этом предполагается, что пользователь не имеет права редактировать исходную топооснову, но может вносить изменения в созданные им самим специализированные слои. При необходимости включения в классификатор карты новых типов объектов, такой классификатор должен быть создан или изменен средствами ГИС Панорама.

Предполагается, что при наличии в таблице адресов, характеризующих расположение объектов, пользователь будет находить фрагмент карты при помощи геокодирования, а затем выбирать уже имеющийся объект или создавать новый на специализированном слое и связывать этот объект с записью таблицы.

Выводы

Разработанная инструментальная система обеспечивает автоматизацию всех функций разработки спецификаций и тем самым позволяет существенно ускорить создание приложений БД. Готовое предметное ПБД создается динамически (без перекомпиляции) в результате интерпретации спецификации в той же инструментальной системе, что позволяет оценивать адекватность создаваемой спецификации.

Предложенный программный интерфейс для взаимодействия с внешними модулями обеспечивает необходимый доступ к наборам данных ПБД и позволяет реализовывать специфические вычисления и обработку данных без изменения кода ПБД.

Картографический модуль реализует взаимодействие ПБД с ПД, а само инструментальное средство «Интерфейс АП» применяется сторонними разработчиками для реализации ГИС-функциональности при разработке ИС.

Разработанные автором лично подсистемы, модули и функции опубликованы в работах [8, 15,47, 48, 50, 51, 55, 57-59,62]. На разработанные инструментальные средства получены свидетельства о государственной регистрации программ для ЭВМ [12, 14, 16].

Глава 4.Применение инструментальной системы «ГеоАРМ»

4.1.Разработка приложения для БД «Pubs»

Для лучшего понимания эффективности предложенной в работе технологии рассмотрено её применение на модельном примере – разработка приложения для БД «Pubs» [86], входящей в качестве примера в состав MS SQL Server и хорошо знакомой многим программистам. Данная БД имитирует хранилище издательской компании и состоит из 11 таблиц: *authors* (авторы), *discounts* (скидки), *employee* (служащие), *jobs* (должности), *pub_info* (информация об издательствах), *publishers* (издательства), *roysched* (авторские гонорары), *sales* (продажи), *stores* (магазины), *titleauthor* (авторы книг), *titles* (книги) (Рисунок **23**).

Рисунок 23. Диаграмма БД «Pubs»

Опираясь на информацию о структуре БД «Pubs», можно выделить следующие задачи автоматизации: Ведение реестра издательств, Ведение реестра изданий и Учёт продаж. Каждая из задач предполагает ведение соответствующих реестров («Издательства», «Издания», «Продажи») и связанных с ними справочников. ПБД «Pubs» должно обеспечивать выполнение CRUD-операций для всех справочников и реестров, а также поддерживать построение пользовательских запросов и создание отчётов. В интерфейсе ПБД для каждой решаемой задачи визуально должны быть выделены подсистемы.

Разработка ПБД «Pubs» в инструментальной системе «ГеоАРМ».

На первом этапе средствами инструментальной системы было создано подключение к БД в технологии ADO. Строка соединения в спецификации:

```
ADO ConnectionString=Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=pubs;Data Source=FEREFEROV
```

Подключение к БД обеспечивает доступ к метаинформации о структуре таблиц, ключах и связях предоставляемых СУБД. Интерпретация связей с точки зрения пользователя позволяет создавать удобный полнофункциональный пользовательский интерфейс. Анализ схемы БД «Pubs» показывает, что между таблицами существует 10 связей различного вида. Связи вида LR (глава 2) извлекаются автоматически из метаданных схемы БД и проставляются таблицам в спецификации. Далее, при необходимости, связи уточняются (например, определяются связи вида DR).

Связь между таблицами *publishers* и *pub_info* типа «один-к-одному» то есть вида PR. Работа с этими таблицами для пользователя должна выглядеть как с одним набором данных, при этом в момент заполнения полей таблицы *pub_info* автоматически проставляется первичный ключ соответствующей записи из таблицы publishers. Для реализации редактирования записей этих таблиц как одного набора данных было создано представление *vPublishers*.

Остальные 9 связей типа «один ко многим», при этом таблицы *titleauthor* и *sales* реализуют связь типа «многие-ко-многим». Соответственно записи из

таблиц *titleauthor* и *sales* формируют «Детали» для каждой записи из связанных с ними таблиц, причём в обе стороны. Для автоматического формирования списка авторов для каждой книги (записи из таблицы *title*) и списка книг для каждого автора (записи из таблицы *authors*) необходима информация о наличии у этих таблиц связи вида DR. Аналогичная информация необходима для формирования списка магазинов, в которых были осуществлены продажи конкретной книги и списка книг, проданных в конкретном магазине. Такая связь необходима для просмотра списка сотрудников конкретного издательства. Средствами «ГеоАРМ» были настроены следующие связи вида DR:

```
//для authors
REFS (   Книги=<vAuthorTitle.au_id )
    …
//для stores
REFS (   ‘Проданные книги’=<vSales.stor_id )
    …
//для title
REFS (   Авторы=<vTitleAuthor.title_id,
         Продажи=<vSales.title_id,
         ‘Гонорары при продаже’=<roysched.title_id )
    …
```

Связи таблицы *publishers* с таблицами *titles* и *employee* говорит о том, что она является справочником для них. С другой стороны, при работе с информацией об издательствах удобно иметь возможность просматривать изданные в нём книги и работающих сотрудников. Для этого необходимо добавить информацию о том, что данные связи для таблицы *publishers* должны иметь вид DR и обеспечивать работу с «Деталями» «Сотрудники» и «Изданные книги»:

```
//для  publishers
REFS (   Сотрудники=<vEmployee.pub_id
         Изданные книги=<vTitles.pub_id …)
```

После уточнения вида связей между таблицами при помощи инструментальной системы «ГеоАРМ» были построены 6 представлений: «Сотрудники» (Рисунок **24**), «Издательства», «Книги», «Авторы книг», «Книги автора», «Продажи». Представления обеспечивают декодирование значений внешних ключей в соответствующие значения записей из подчиненных таблиц (справочников), позволяя работать со связанными таблицами как с одним набором данных. При этом пользователь может выбирать значения из справочников при помощи простых выпадающих списков (ComboBox) и выпадающих списков с функцией автокомплит.

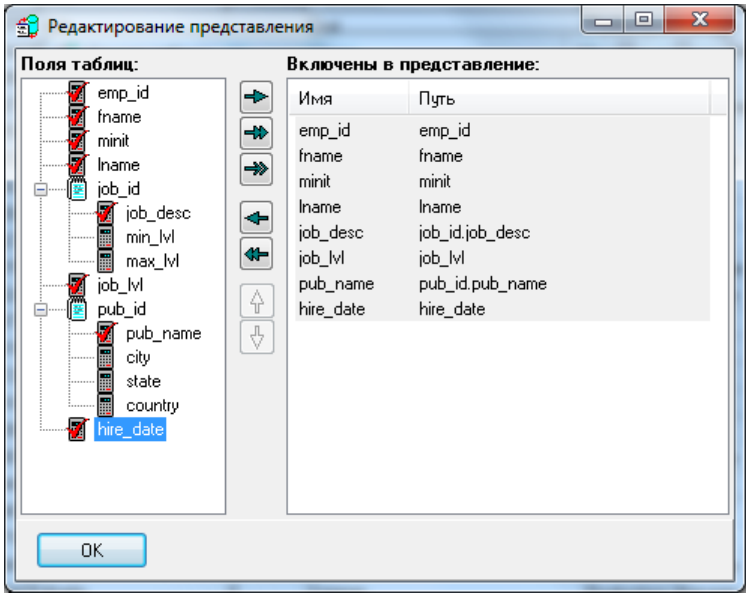

Рисунок 24. Форма редактирования структуры представления "Сотрудники"

На заключительном этапе создания приложения БД «Pubs» было настроено дерево сущностей – элемент пользовательского интерфейса для выбора таблиц и представлений. В дереве сущностей по числу задач решаемых системой были выделены три раздела: «Издательства», «Издания», «Продажи». Раздел «Издательства» содержит представления «Издательства» (*vPpublishers*), «Сотрудники» (*vEmployee*) и справочник «Должности» (*jobs*). В раздел «Продажи» вошли представление «Продажи» (*vSales*) и таблицы «Магазины» (*stores*) и «Скидки» (*discounts*). Представление «Издания» (*vTitles*), таблицы «Авторы» (*authors*) и «Гонорары» (*roysched*) вошли в состав раздела «Издания».

Настройка приложения БД «Pubs» (Рисунок **25**) заняла 1 час, объём спецификации системы составил 20 предложений на языке ЯПБД, 160 строк.

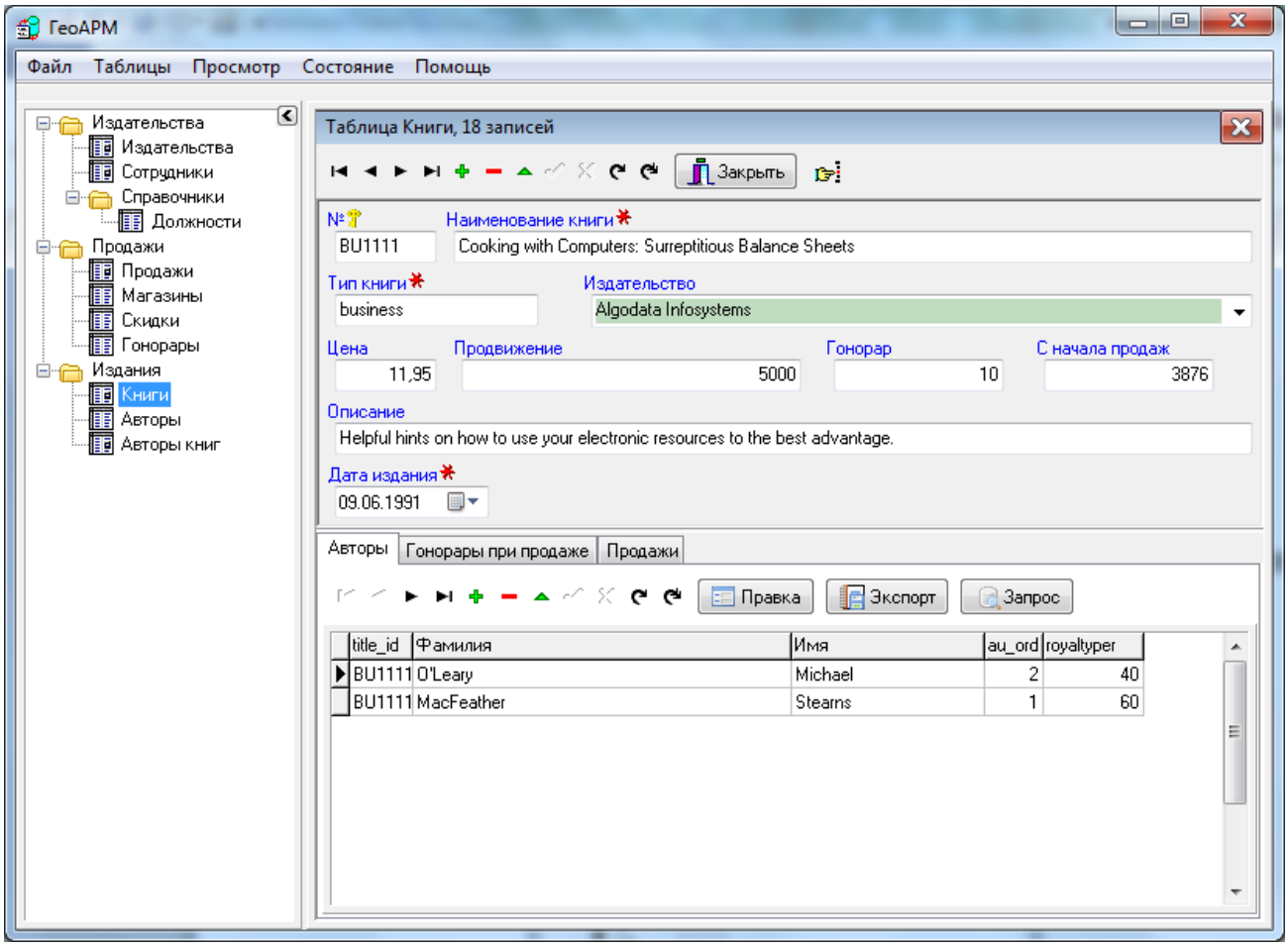

Рисунок 25. Приложение БД "Pubs"

Разработка ПБД «Pubs» в MS Access. Для взаимодействия с внешними БД в MS Access существует несколько подходов. Наиболее подходящим для решаемой задачи является создание проекта (Microsoft Access Project), позволяющего организовать непосредственное взаимодействие с предметными БД MS SQLServer (без импорта объектов БД в MS Access). На первом этапе также, как и при использовании «ГеоАРМ», создается соединение с БД, при этом выбрать поставщика данных не представляется возможным, поскольку данный механизм рассчитан на взаимодействие исключительно с MS SQLServer. После подключения к предметной БД Access обеспечивает просмотр свойств таблиц и представлений.

Следующим шагом является уточнение связей и создание представлений. В режиме конструктора таблиц существует возможность настройки подстановок для полей, являющихся FK (связанных с другими таблицами), что обеспечивает выбор значений из справочников при помощи выпадающих списков. При этом существует возможность включить в выпадающий список несколько полей из справочника, но в целевой таблице всегда отображается только одно поле подстановки из справочника, что является существенным минусом. Данный механизм был применен для простых справочников таких, как *jobs, publishers* для таблицы *employee*, а также *publishers* для *titles.* В остальных случаях для реализации отображения нескольких полей из подчиненных таблиц на основе целевых таблиц были созданы представления, содержащие несколько полей-подстановок из справочников. На основе таблицы *titleauthor* было создано два представления («КнигиАвторов», «АвторыКниг»), в первом случае содержащее поля-подстановки из таблицы *titles,* во втором – из *authors*. На основе *sales* было создано представление «Продажи», включающее атрибуты stor_name, stor_addres, city из справочников *stores* и *title*, *type*, *pub_name* из *title* (поле *pub_name* является подстановкой из *publishers*).

Для таблиц, на PK которых существуют ссылки из других таблиц, существует механизм указания «таблиц-деталей» (или «представлений-деталей»). Однако данный механизм рассчитан на описание взаимодействия только с одним набором данных. Так, для таблицы *titles* не представляется возможным указать связи со всеми существующими «таблицами-деталями» (*titleauthor, roysched, sales*). Были указаны следующие связи с «таблицами-деталями»: для *jobs* и *publishers* таблица *employee*, для *authors* представление *КнигиАвторов*, для *stores* представление *«Продажи»*, для *titles «АвторыКниг»*.

После настройки всех наборов данных и связей между ними следует этап создания пользовательских форм. В MS Acceess реализована автоматическая генерация пользовательских форм с применением шаблонов – «в один столбец» и «выровненный» для форм редактирования индивидуальных записей, «ленточный» и «табличный» для форм взаимодействия с наборами данных (таблиц,

представлений). Если определены связи с «таблицами-деталями», т.е. возможность генерации «подчиненных форм» – форм, реализующих взаимодействие с индивидуальной записью и соответствующими записями «таблиц-деталей». Как правило, сгенерированные формы требуют доработки: реализации переходов между формами, более рационального размещения компонентов на форме, реализации реакций на изменения значений из справочников. Всего для таблиц и представлений было сгенерировано и доработано 18 форм. Необходимо отметить, что на форме для таблицы *titles* были встроены дополнительные компоненты для взаимодействия со всеми «таблицами-деталями», так как механизмы MS Access позволяют настроить связь и сгенерировать форму для взаимодействия только с одной.

Разработка приложения БД «Pubs» средствами MS Access заняла 7 часов. Несмотря на наличие в MS Access развитых средств настройки связей и генерации форм в процессе разработки ПБД потребовалось написание и отладка программного кода. В сравнении с разработкой ПБД в ГеоАРМ (рисунок 26), в MS Access наибольших усилий и времени потребовал этап разработки пользовательских форм. Так, при разработке форм для таблиц (представлений), у которых существует больше одной «таблицы-детали», пришлось непосредственно реализовывать взаимодействие с ними, что в ГеоАРМ решается гораздо проще настройкой связей вида DR. Очевидным преимуществом применения ГеоАРМ по сравнению с MS Access стало отсутствие необходимости разработки форм.

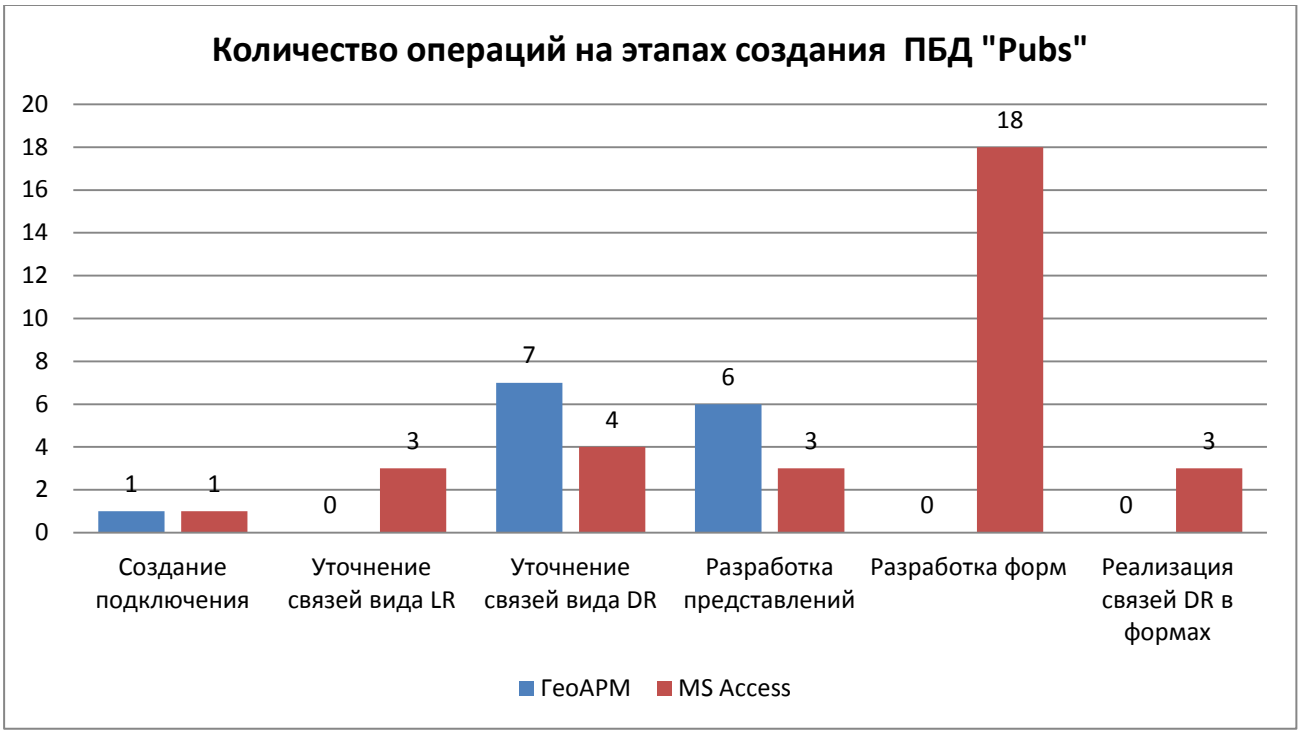

Рисунок 26. Сравнение разработки ПБД в ГеоАРМ и MS Access.

4.2. Разработка АИС «Управление многоквартирными домами г. Иркутска»

Требования к разработке. Целью создания информационной системы «Управление многоквартирными домами г. Иркутска» (АИС УМД) – обеспечение органов местного самоуправления, физических и юридических лиц достоверными сведениями, повышение эффективности подготовки и обоснованности принятия управленческих решений администрации г. Иркутска по вопросам управления жилищно-коммунальным хозяйством, в частности, управления многоквартирными домами (МД), расположенными на территории г. Иркутска. Создание АИС УМД было нацелено на решение следующих задач:

- создание общего банка достоверной информации в виде тематических баз данных, обеспечивающего обработку, надежное централизованное хранение и оперативный поиск электронных документов и информации, связанной с управлением многоквартирными домами органов местного самоуправления г. Иркутска, физических и юридических лиц;

- автоматизация процессов сбора, обработки, представления информации в сфере управления многоквартирными домами;
- автоматизация процессов подготовки документации и учета результатов проведения общих собраний в многоквартирных домах по вопросам определения способа управления МД и выбора управляющей организации;
- автоматизация процессов подготовки конкурсной документации для проведения открытых конкурсов по выбору управляющих организаций;
- автоматизация расчета платы за пользование жилым помещением для нанимателей по договорам социального найма;
- автоматизация расчета оставшейся части платы за содержание и ремонт жилого помещения в случаях, когда размер вносимой нанимателями муниципального жилищного фонда г. Иркутска по договорам социального найма жилых помещений и договорам найма жилых помещений платы ниже, чем размер платы, установленный договорами управления;
- улучшение качества обработки тематической информации, повышение оперативности обмена информацией, сокращение непроизводительных операций по поиску, обработке и оформлению жилищно-коммунальной документации;
- обеспечение контроля за деятельностью организаций, управляющих многоквартирными домами;
- оперативное представление информации в сфере жилищно-коммунального хозяйства вышестоящим органам и смежным подразделениям.

Технические требования на создание АИС УМД регламентировали создание многопользовательской информационной системы, работающей под ОС семейства MS Windows, обладающей возможностью гибкой модернизации в условиях меняющегося законодательства в сфере ЖКХ и обеспечивающей информационный обмен со смежными прикладными программными системами подразделений администрации г. Иркутска.

Исходя из технических возможностей информационно-вычислительной инфраструктуры администрации г. Иркутска было принято решение разработать

информационную систему с клиент-серверной архитектурой. В качестве СУБД была выбрана MS SQL Server, как одна из самых мощных по производительности, обеспечивающая надежное хранение и оперативный поиск информации в БД. Кроме того, MS SQL Server 2008 обладает большими возможностями по организации информационного обмена с различными СУБД.

В процессе разработки системы УМД требовалось уточнение требований на основе мнений пользователей и поэтапное расширение перечня атрибутов, наращивание функционала, а также подключение данных из смежных подсистем. Поэтому для разработки была выбрана технология создания АИС на основе декларативных спецификаций, как отвечающая требованиям методологий инкрементальной спиральной разработки и не требующей переработки и компиляции программного кода.

Взаимодействие со смежными системами. АИС «Управления многоквартирными домами» имеет достаточно сложную структуру. По составу информации и решаемым задачам в АИС УМД условно выделены шесть подсистем: «Многоквартирные дома», «Обеспечение конкурсов по выбору управляющей организации», «Муниципальная собственность в МД», «Управляющие организации», «Капитальный ремонт», «Земельные участки МД». База данных АИС УМД насчитывает порядка 90 таблиц и представлений (View). Часть информации в АИС УМД поступает из смежных систем, функционирующих в подразделениях администрации г. Иркутска (рисунок 27). Рассмотрим межсистемные информационные потоки (таблица 10).

Таблица 10. Информационные потоки АИС УМД со смежными системами

	ИС «Управление многоквартирными домами» (Комитет по жилищно-коммунальному хозяйству г. Иркутска. СУБД – MS SQL Server)	
	Получает	Предоставляет
ИС «Единый общегородской регистр	Адресные справочники (Улицы, Типы улиц,	Информация об организации, управляющей

адресов недвижимости» (Департамент по градостроительной политике г. Иркутска. СУБД – MS SQL Server)	Микрорайоны, Районы, Округа, Адреса), реестр объектов недвижимости.	МД.
ИС «Бюро технической инвентаризации» (МУП БТИ г. Иркутска. СУБД – Oracle)	Технические характеристики зданий МД.	Информация об организации, управляющей МД.
ИС «Объекты муниципальной собственности г. Иркутска» (Комитет муниципального имущества г. Иркутска. СУБД – MS SQL Server)	Информация о наличии помещений в МД, находящихся в муниципальной собственности с техническими характеристиками.	
МГИС г. Иркутска	Цифровой Адресный план г. Иркутска	Слой «Придомовые территории МД»

ИС «Единый общегородской регистр адресов недвижимости» ЕОРАН является базовым информационным ресурсом, содержащим адресную информацию об объектах хозяйственной деятельности администрации г. Иркутска, которая регулярно актуализируется на основе документов о вводе зданий в эксплуатацию и присвоению им адресов. Интеграция с данной информационной системой избавляет от необходимости ведения адресных справочников в разрабатываемой системе и упрощает межсистемную интеграцию со смежными подсистемами, использующими ту же адресную инфраструктуру. Для взаимодействия АИС УМД с АИС ЕОРАН средствами MS SQL Server созданы связи (Link), через которые обеспечивается доступ на чтение к таблицам

адресных справочников для АИС УМД и к представлению «Многоквартирные дома» в БД УМД для АИС ЕОРАН.

Взаимодействие с АИС «Бюро технической инвентаризации» (АИС БТИ) обеспечивает поступление в АИС УМД технических характеристик зданий и помещений многоквартирных домов. Такая информация позволяет автоматизировать подготовку документации для проведения мероприятий (собрания, конкурсы) по определению способа управления и организации, управляющей МД, а также позволяет автоматизировать расчеты средств, необходимых для капитальных ремонтов МД. Поступление данных из АИС БТИ реализовано через DTS-пакеты (Data Transformation Services), что обусловлено необходимостью взаимодействия разнородных СУБД (Oracle и MS SQL Server), а также необходимостью трансформации поступающих данных в целевую структуру БД УМД.

Информация, поступающая из АИС «Объекты муниципальной собственности г. Иркутска» позволяет обеспечить сотрудников, работающих с АИС УМД, данными о наличии помещений в МД, находящихся в муниципальной собственности, что позволяет автоматизировать расчеты компенсаций по оплате содержания данных помещений из бюджета города, подготовку документов для представления интересов города Иркутска при проведении мероприятий по определению способа и организаций, управляющих МД. Межсистемное взаимодействие организовано на уровне СУБД, созданы связи (Link) между серверами (MS SQL Server) и разработан DTS-пакет для трансформации данных из БД «Объекты муниципальной собственности г. Иркутска» в структуру БД УМД. Трансформация данных необходима в связи с отличием структуры адресных справочников, используемых в системах.

Муниципальная геоинформационная система (МГИС) г. Иркутска содержит пространственно-распределенные данные об объектах городского хозяйства в виде базы данных тематических цифровых карт (слоев) в формате ГИС «Карта 2008» (Панорама). МГИС г. Иркутска содержит большое количество слоев (например, «Электросети», «Сети тепло-, водоснабжения», «Ветхое жилье»,

«Объекты культурного наследия», «Избирательные участки»), базовым из которых является «Адресный план», содержащий метрическую (пространственную) и семантическую информацию о зданиях и дорогах г. Иркутска. Доступ к цифровым картам осуществляется через GIS Server 2011. Для решения задач управления многоквартирными домами был создан специальный слой «Многоквартирные дома», работа с которым была реализована через подсистему «Карта» (глава 3). Взаимодействие с МГИС г. Иркутска позволяет решать задачи учета придомовых территорий МД, определение зон уборки придомовых территорий, определения точек ввода сетей снабжения в МД, принятия решений по установке детских и спортивных площадок.

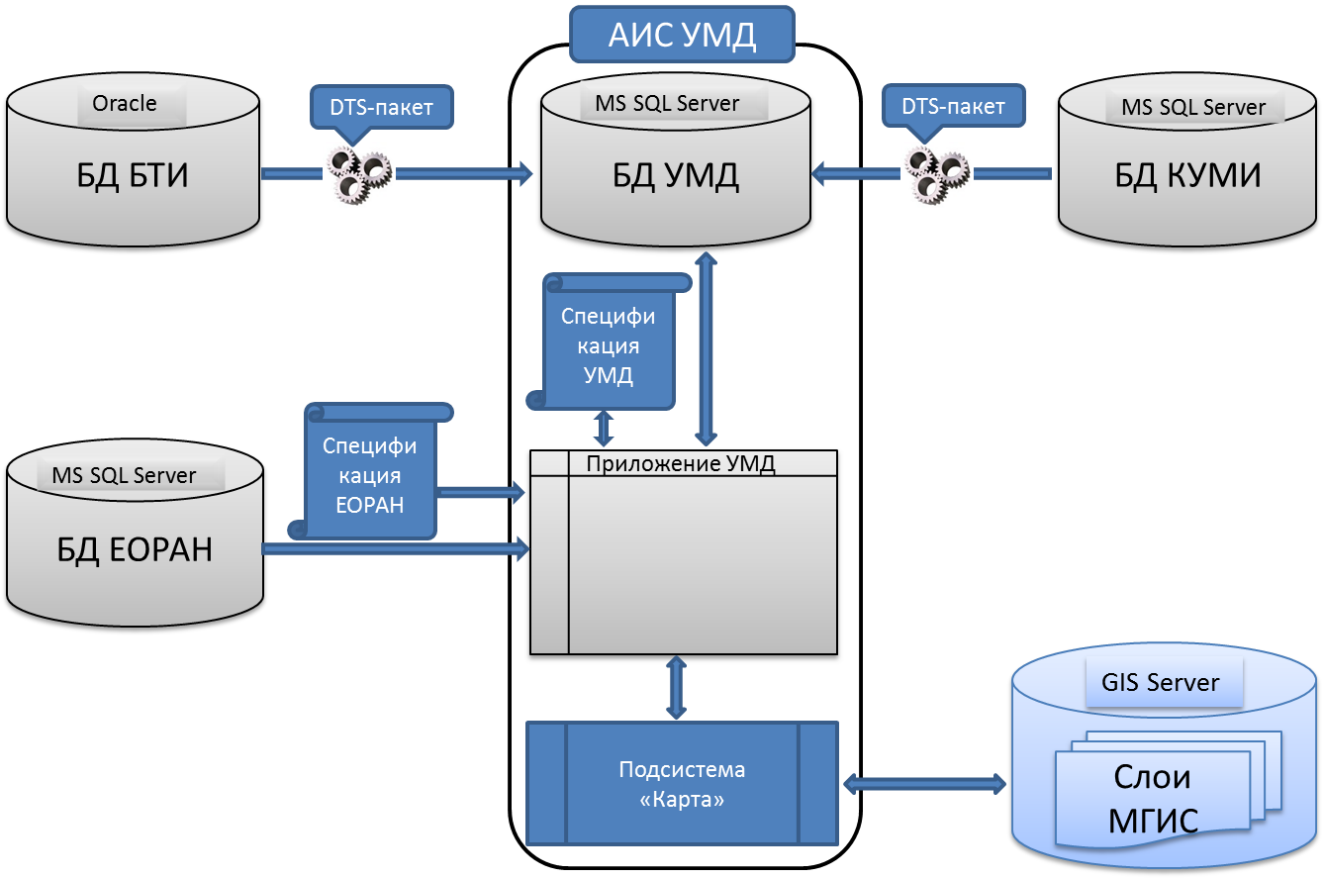

Рисунок 27. Взаимодействие АИС УМД со смежными системами

Программное обеспечение АИС УМД

Подсистема «Многоквартирные дома» обеспечивает формирование МД, как объектов учёта жилищно-коммунального хозяйства, ведение технической

информации и показателей по многоквартирным домам и их помещениям, а также учёт выбранных способов управления и организаций, управляющих МД.

Основными объектами информатизации в АИС УМД являются многоквартирные дома. В качестве идентификатора МД выступает его адрес: «Округ», «Улица», «Тип улицы», «Дом», «Литера строения». В целом адресная информация является системообразующей во всех муниципальных информационных системах. Для обеспечения адресной инфраструктуры в АИС УМД используются адресные справочники БД ЕОРАН со следующей структурой (Рисунок **28**)

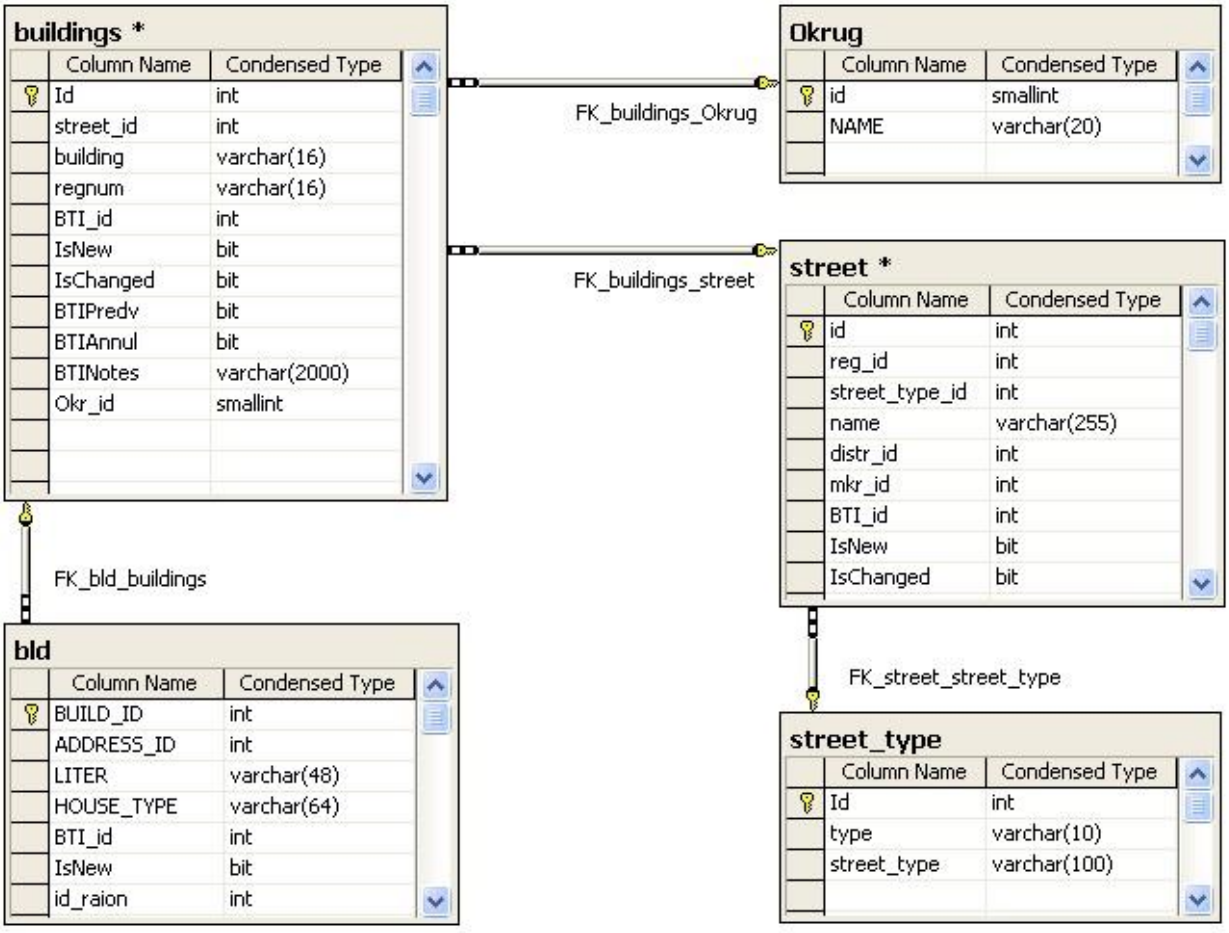

Рисунок 28. Структура адресных справочников БД ЕОРАН

АИС ЕОРАН разработана в технологии создания информационных систем на основе декларативных спецификаций, что позволяет ссылаться на готовые описания таблиц и представлений АИС ЕОРАН, указав ссылку в спецификации: `USES ADDR=('EORAN.dbpl', 'ADDR.dbo.')`. Для работы с адресами зданий в АИС УМД используется представление vAddress:

```
VIEW vAddress ON Address AS ‘Адреса’
    (Id=№,
     BTI_id= AS ’Код в БТИ’,
     okrid=OKR.okrug_ID AS -,
     okr=OKR.okrug_ID.name AS Округ,
     strt=street_id.ST AS Тип,
     strn=street_id.name AS Улица,
     mk= street_id.mk AS Микрорайон,
     building= AS ’Номер дома’)
NAMES=strn;strt;building
```

Поскольку по одному адресу могут находиться несколько строений (жилой дом, пристрой, магазин), для идентификации конкретных зданий почтового адреса недостаточно, поэтому в градостроительном хозяйстве принято к почтовому адресу добавлять строительный литер здания. Многоквартирные дома могут состоять как из одного здания, так и нескольких. Кроме того, по одному почтовому адресу могут находиться несколько зданий, которые имеют отдельные точки подключения к коммуникациям, и, следовательно являться независимыми МД. Поэтому для идентификации многоквартирного дома почтовый адрес дополняется Литером, который в случае, если МД является одним зданием совпадает с Литером здания, в противном случае задаётся оператором системы.

Для организации формирования и учёта информации о многоквартирных домах, их технических характеристик, а также помещений МД разработана следующая структура в БД УМД (Рисунок **29**):

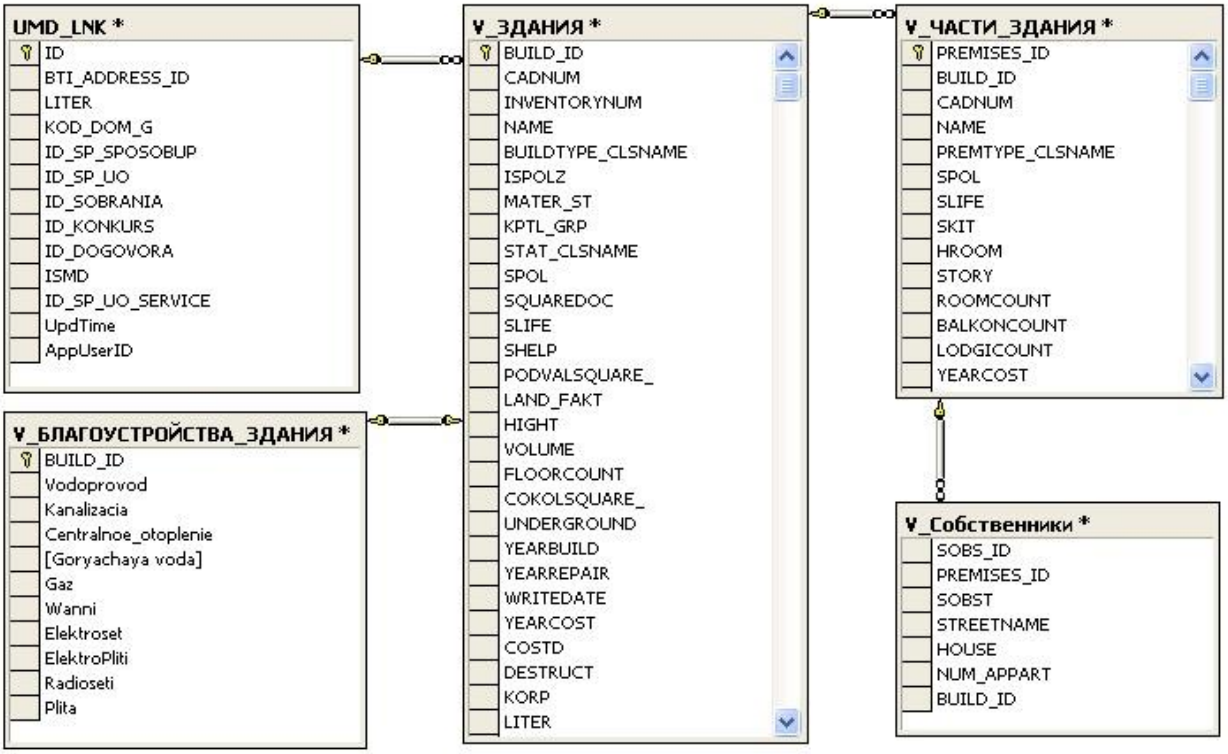

Рисунок 29. Таблицы подсистемы "Многоквартирные дома"

Таблицы V_Здания, V_Благоустройства_Здания, V_Части_Здания и V_Собственники являются преобразованными аналогами таблиц БД БТИ, данные в которых обновляются раз в неделю через DTS-пакет. Таблица V_Здания содержит информацию о всех жилых зданиях (в том числе и частных) г. Иркутска с их техническими характеристиками. В таблице V_Благоустройства_Здания, связанной «один-к-одному» с V_Здания, содержится информация о наличии средств благоустройства в зданиях (холодная, горячая вода, отопление, газ, электроплиты, ванны). Техническая информация о квартирах в зданиях содержится в таблице V_Части_Здания, а об их собственниках в V_Собственники.

Таблица UMD_LNK реализует формирование информационного объекта «Многоквартирный дом» и обеспечивает учет последних статусов МД – ссылки на последнее собрание, последний конкурс, текущий выбранный способ управления, текущую управляющую организацию и действующий договор с ней. В БД УМД на основе V_Здания и V_Части_Здания создано View MD_PROPERTY, агрегирующее количественные характеристики по МД, которое связано средствами спецификации с UMD_LNK связью вида PR.

Данные из таблицы V_Здания для записей из UMD_LNK реализуют детали, что обеспечивает формирование объектов многоквартирных домов, поэтому данная связь для UMD_LNK в спецификации описывается как DR. Описание на ЯПБД для таблицы UMD_LNK выглядит следующим образом:

```
TABEL UMD_LNK AS 'Многоквартирные дома' (
     ID=P,
     BTI_ADDRESS_ID=^ADDR.vAddress.BTI_id, //ссылка на адрес
     LITER=S, //Литер МД
     ID_SP_SPOSOBUP=^SP_SPOSOBUP,//ссылка на способ управления
     ID_SP_UO=^vSP_UO,//ссылка на управляющую организацию
     ID_SP_UO_SERVICE=^vSP_UO,//ссылка на обслуживающую организацию
     ID_SOBRANIA=^SOBRANIA,//ссылка на последнее собрание
     ID_KONKURS=^KONKURS//ссылка на последний конкурс
      )
     //описание связей с таблицами деталей
REFS=(Здания=<~vBUILDINGS.UMD_LNK_ID
     Договоры=<vDOGOVOR.UMD_LNK_ID
     Собрания=<vSOBRANIA.UMD_LNK_ID
     Конкурсы=<vKONKURS.UMD_LNK_ID
     TH=ID^MD_PROPERTY(UMD_LNK_ID)   //связь   вида   PR   с   таблицей
MD_PROPERTY
```

Для пользователей создано представление vUMD_LNK («Многоквартирные дома»), через которое они осуществляют просмотр информации по многоквартирным домам и формирование новых информационных объектов МД. Описание представления vUMD_LNK:

```
DISPLAY vUMD_LNK on UMD_LNK AS 'Многоквартирные дома' (
     ID= AS -
     [GB:Адрес
          OK=BTI_ADDRESS_ID.OKR AS Округ
          STR=BTI_ADDRESS_ID.STRN AS 'Тип ул.'
          TSTR=BTI_ADDRESS_ID.strt AS Улица
          D=BTI_ADDRESS_ID.building AS Дом
          LITER= AS Литер МД
     ]
```

```
    [GB:'Управление и обслуживание'
        SU=ID_SP_SPOSOBUP.Name AS 'Способ управления'
        UO_Name=ID_SP_UO.SName AS 'Управляющая организация'
        SERVICE_ORG=ID_SP_UO_SERVICE.SName AS 'Обслуживающая
        органихация'
    ]
    [GB:'Количественная характеристика'
        COL_FLAT=TH.FL_CNT AS 'Кол-во квартир'
        ALL_S=TH.ALL_S AS 'Общая площадь'
        LIVE_S=TH.LIV_S AS 'Жилая площадь'
    ]
    [GB:'Количественная характеристика помещений'
        CLP=CNA.CNT_LP AS 'Кол-во жилых'
        ASLP=CNA.AS_LP AS 'Общая площадь жилых'
        LSLP=CNA.LS_LP AS 'Жилая площадь жилых'
        CNP=CNA.CNT_NP AS 'Кол-во нежилых'
        ASNP=CNA.AS_NP AS 'Общая площадь нежилых'
        CLPM=CNA.CNT_LPM AS 'Кол-во муниципальных жилых'
        ASLPM=CNA.AS_LPM AS 'Общая площадь муниципальных жилых'
        LSLPM=CNA.LS_LPM AS 'Жилая площадь муниципальных жилых'
        CNPM=CNA.CNT_NPM AS 'Кол-во муниципальных нежилых'
        ASNPM=CNA.AS_NPM AS 'Общая площадь муниципальных нежилых'
    ]
```

Разработанное представление vUMD_LNK позволяет автоматически формировать пользовательский интерфейс (Рисунок 30) обеспечивающий оперативное получение работниками КЖКХ количественных и качественных показателей по каждому многоквартирному, а также формировать новые информационные объекты «Многоквартирные дома».

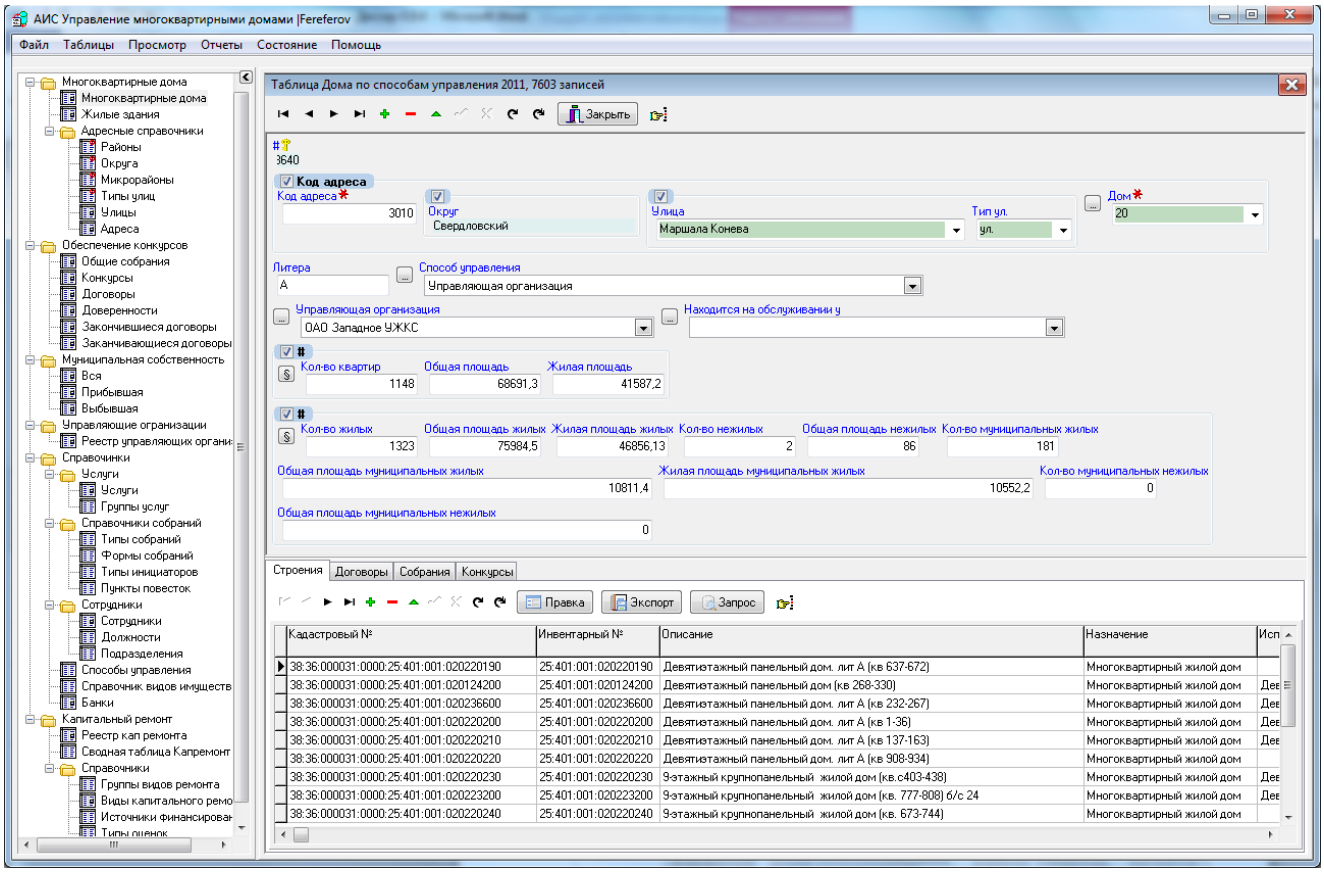

Рисунок 30. Подсистема МД

Для каждой записи многоквартирного дома можно просматривать информацию о текущем способе управления, управляющей организации (если она есть), площадных и количественных характеристиках, всех зданиях, входящих в состав МД, договорах, собраниях и конкурсах, касающихся данного МД.

Подсистема «Обеспечение конкурсов по выбору управляющей организации» предназначена для решения задач сопровождения и обеспечения мероприятий по определению способа управления и организации управляющей многоквартирным домом, а также учета результатов этих мероприятий. Согласно ст.161 ЖК РФ жильцы каждого многоквартирного дома в первую очередь должны выбрать способ управления МД. Существуют три способа управления МД: Непосредственное управление собственниками (НУС), Товарищество Собственников Жилья (ТСЖ) и Управляющая организация (УО). Способ управления определяется большинством голосов на общем собрании собственников помещений многоквартирного дома. Далее, в зависимости от выбранного способа управления МД собственники должны заключить договоры с

компаниями – поставщиками ЖКУ непосредственно каждый сам или через управляющую организацию, которую определяют также общим голосованием.

Часто в многоквартирных домах есть помещения, находящиеся в муниципальной собственности. Поэтому администрация города Иркутска как собственник помещений в МД должна принимать участие в мероприятиях по определению способа управления и организации управляющей МД. Для представления интересов муниципального образования (МО) на собраниях назначается ответственный сотрудник КЖКХ и оформляется на него доверенность.

В случае, если собственники помещений в МД не определили способ управления и управляющую организацию в течение года, администрация города Иркутска должна подготовить, объявить и провести муниципальный конкурс по выбору организации, которая будет управлять конкретным МД.

Схема определения способа управления и управляющей организации представлена на рисунке 31.

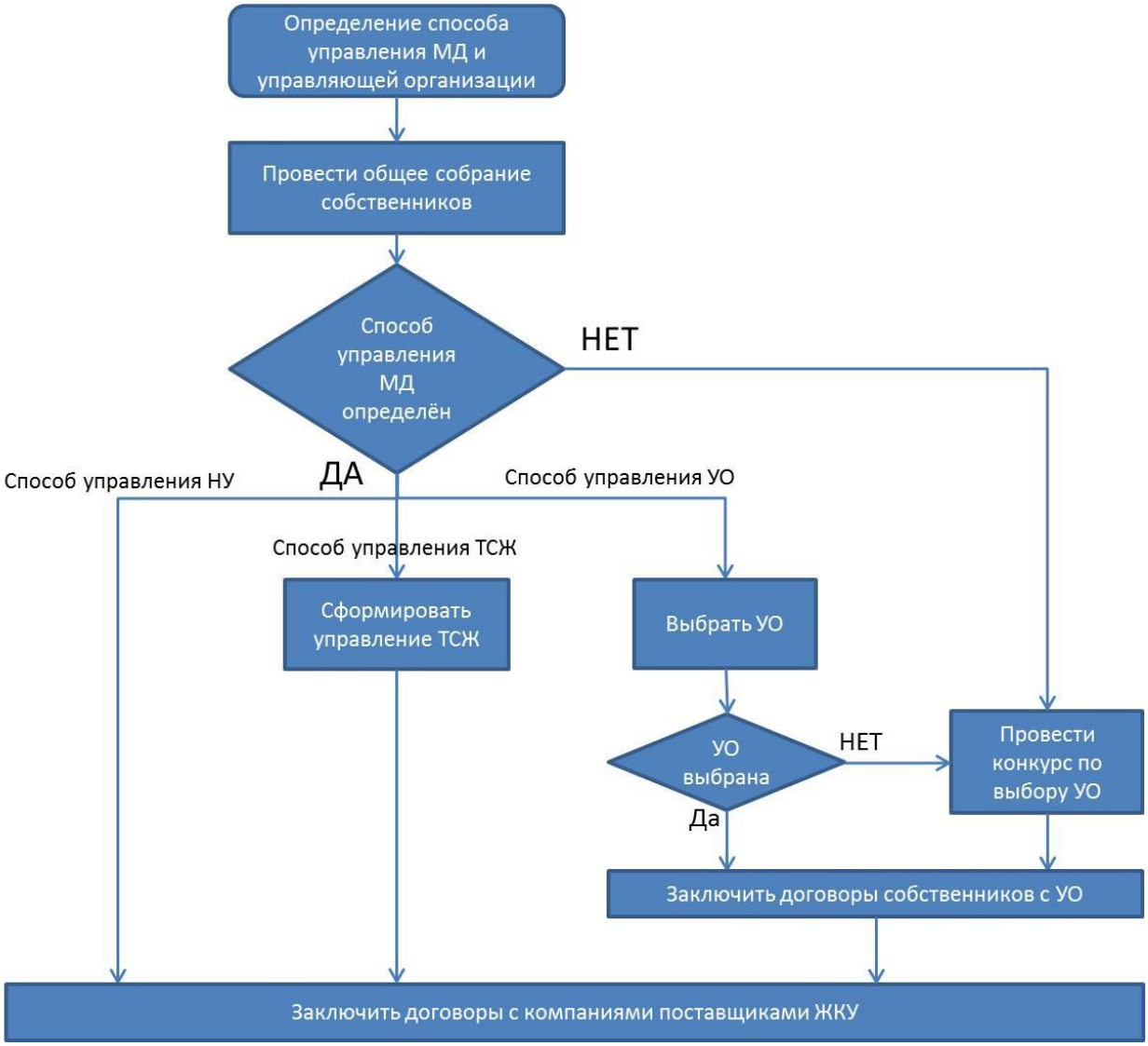

Рисунок 31. Схема определения способа управления

Подсистема «Обеспечение конкурсов по выбору управляющей организации» объединяет функции ведения реестров общих собраний, конкурсов, договоров, заключённых с УО, и доверенностей на представление интересов МО г. Иркутска. В БД УМД создана структура состоящая из 4 списков: «Собрания», «Конкурсы», «Договоры», «Доверенности»; 5 справочников: «Тип собрания», «Форма собрания», «Тип инициатора собрания», «Способ управления», «Управляющая организация» и 2 view (сохранённых запросов) «Заканчивающиеся договоры» и «Закончившиеся договоры».

На основе списков при помощи инструментальной системы «ГеоАРМ» построены представления vSobrania («Собрания»), vKonkurs («Конкурсы»), vDogovors («Договоры»), vDover («Доверенности»).

Реестр выданных доверенностей (Рисунок **32**) на представление интересов МО г. Иркутск на общих собраниях собственников помещений в МД и при проведении конкурсов по выбору управляющих организаций реализован в представлении vDover («Доверенности»). Доверенность оформляется по каждому многоквартирному дому, в котором присутствует муниципальная собственность.

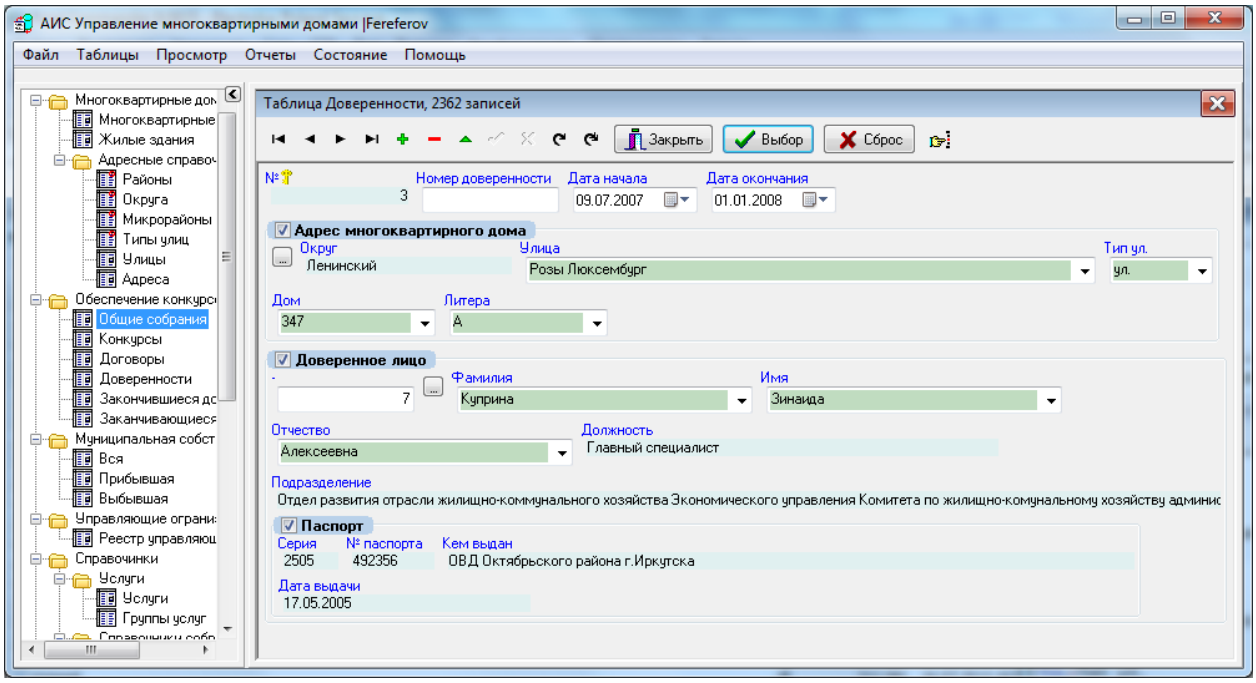

Рисунок 32. Реестр «Доверенностей на представление интересов МО. г. Иркутск»

Представление vSobrania обеспечивает создание пользовательского интерфейса (Рисунок **33**) для выполнения функций учёта проводимых в многоквартирных домах собраний, их повесток, принимаемых на них решений, представителей МО г. Иркутска (ссылка на доверенность) на собраниях и реализации решений.

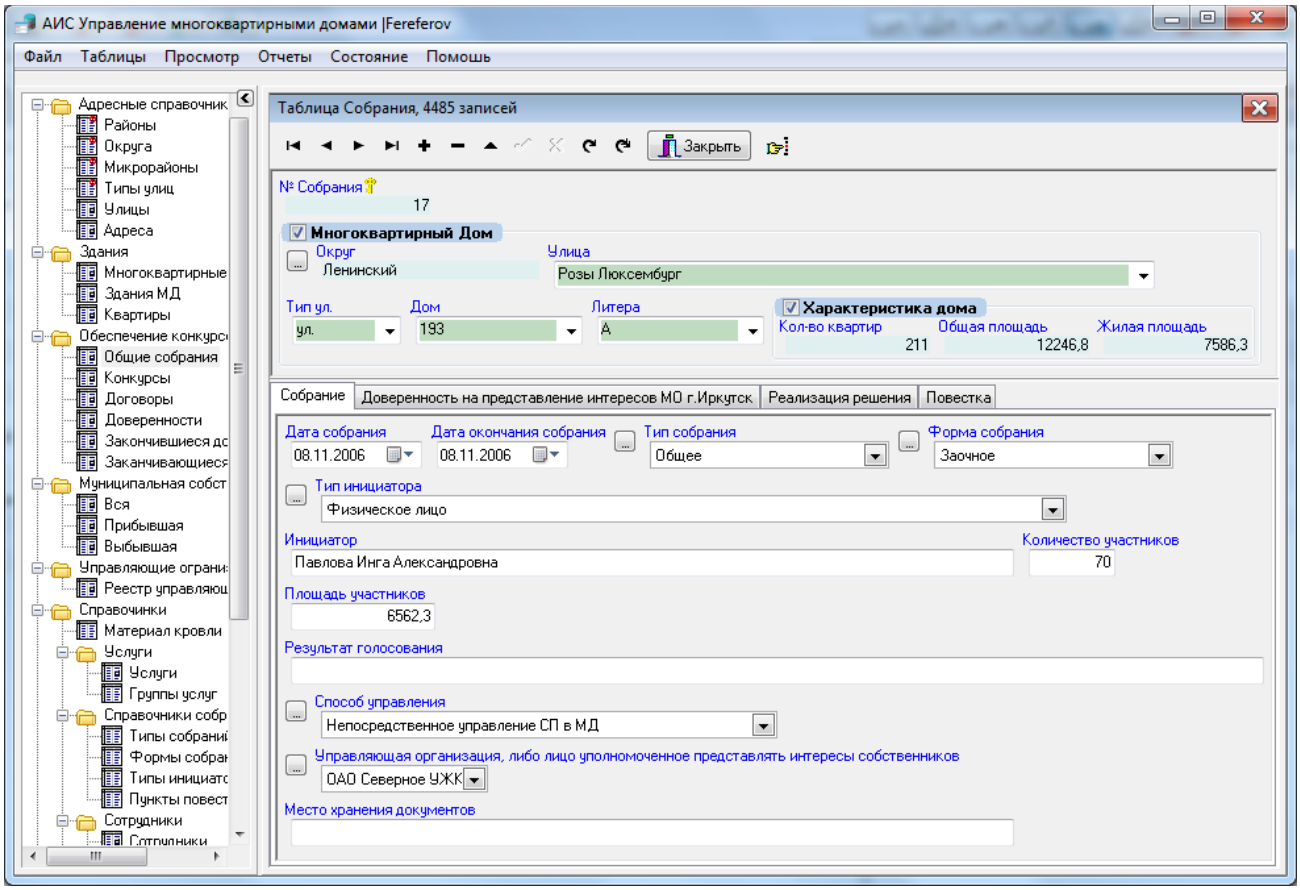

Рисунок 33. Представление "Собрание"

Под реализацией решения понимается наличие реквизитов договора, заключённого с управляющей организацией по результатам собрания. При этом поля реквизитов договора физически относятся к таблице Dogovors. В описании таблицы Sobrania содержится информация о связи вида PR с таблицей Dogovors, что обеспечивает иллюзию работы с полями обеих таблиц, как с одним набором данных, а при заполнении реквизитов договора автоматически создаётся запись в таблице Dogovors со ссылкой на текущую запись таблицы Sobrania.

Представление vKonkurs обеспечивает подготовку документации для проведения открытых конкурсов по выбору управляющих МД организаций и учёт их результатов (Рисунок **34**). При подготовке документации для проведения конкурса по выбору УО в конкурсных требованиях могут быть учтены перечни необходимых обязательных и дополнительных услуг. Данные перечни в представлении vKonkurs реализованы как детали, для чего в описании указаны связи вида DR:

```
      …
REFS=(Перечень обязательных услуг=<vSERV_K.ID_KONKURS,
      Перечень дополнительных услуг=<vDOPSERV_K.ID_KONKURS,
      Участники конкурса=<vUCHASTNIKI.ID_KONKURS)
      …
```

Заполненные конкурсные требования могут быть выгружены в отчёт «Карточка конкурса», реализованного через надстройки (plugins). Кроме того, по результатам проведения конкурса учитываются организации-участники конкурса, УО-победитель и реквизиты договора заключённого с УО-победителем. Список организаций-участников конкурса реализован в виде деталей. Заполнение реквизитов договора реализовано через связь вида PR (аналогично с vSobrania).

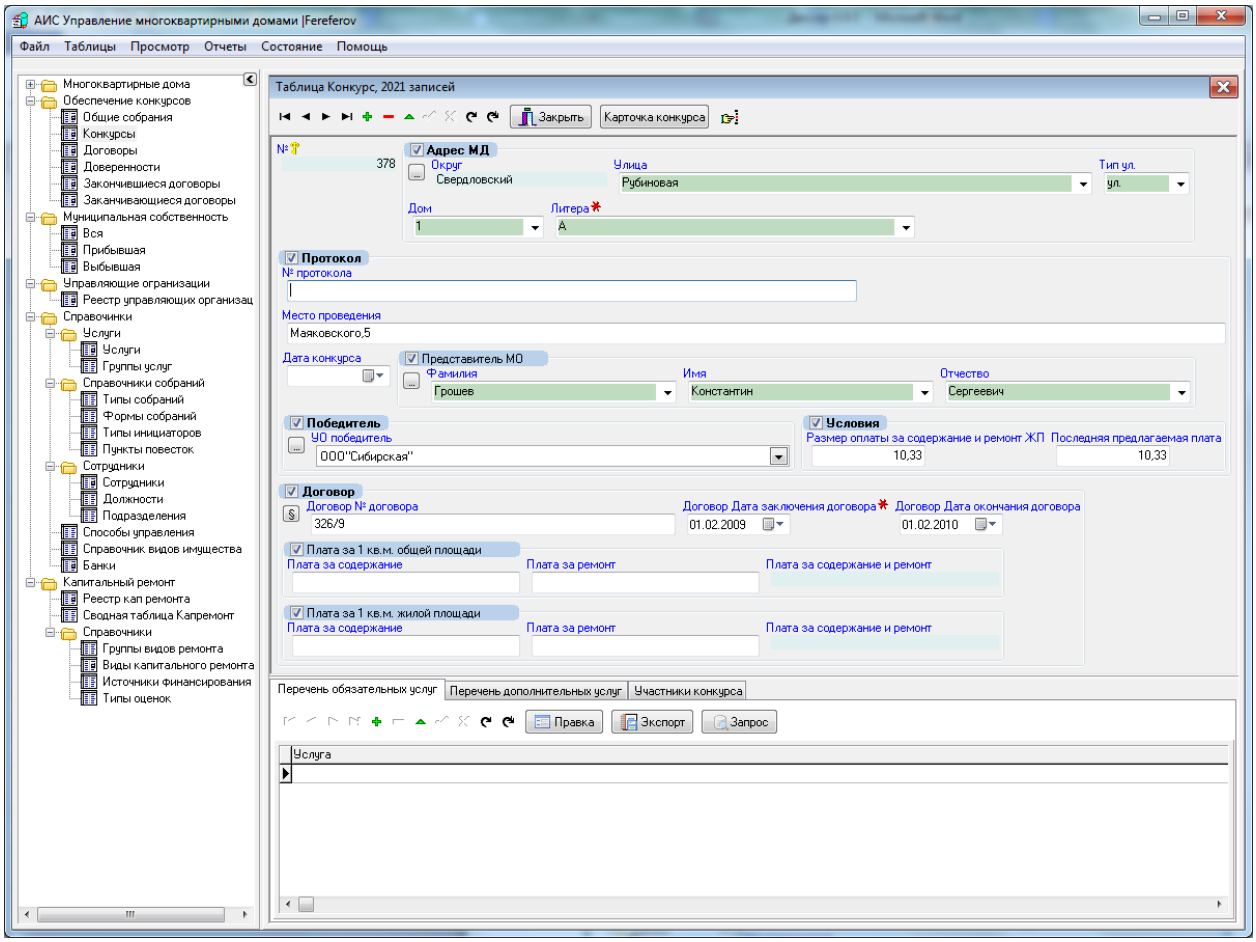

Рисунок 34. Представление "Конкурс по выбору УО"

Подсистема «Муниципальная собственность в МД» предназначена для информационного обеспечения сотрудников КЖКХ информацией о наличии,

состоянии и изменении муниципальной собственности в многоквартирных домах. Данная информация представлена в системе тремя представлениями: vMUNICIPAL_HAVING (Вся муниципальная собственность в МД), vMUNICIPAL_HAVING_INCOME (Прибывшая за последний месяц) и vMUNICIPAL_HAVING_GONE (Выбывшая за последний месяц). Данные представления построены на основе view (сохранённых запросов) реализующих соответствующие выборки поступающих из КУМИ данных.

Представления с информацией о муниципальной собственности могут просматриваться как непосредственно, так и в деталях зданий многоквартирный домов.

Подсистема «Управляющие организации» обеспечивает учёт информации о деятельности управляющих организаций, ТСЖ и других потребительских кооперативов осуществляющих в рамках ЖК РФ деятельность по обслуживанию многоквартирных домов на территории г. Иркутска. Для работы с информацией об управляющих организациях создано представление vSP_UO. Данное представление позволяет работать с организационной информацией УО (наименование, адрес, телефон, ОГРН, ИНН, банковские реквизиты,…), а также просматривать список МД, которыми управляет или обслуживает данная УО (Рисунок **35**).

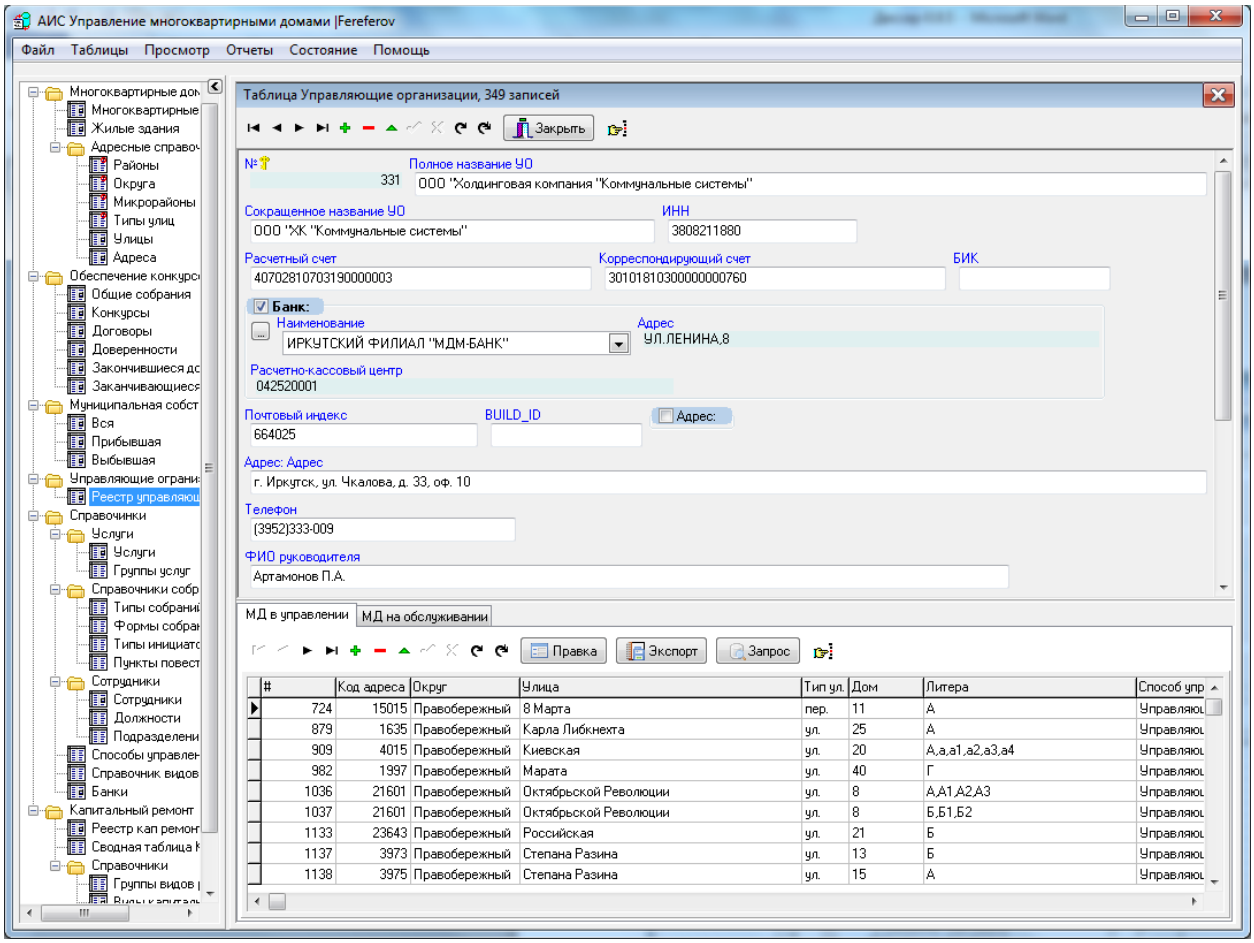

Рисунок 35. Подсистема "Управляющие организации"

Подсистема «Капитальный ремонт» обеспечивает информационное сопровождение планирования мероприятий по капитальному ремонту многоквартирных домов. Подсистема реализована в виде «Реестра капитального ремонта МД» (представление vR_CAPITAL_REPAIR), списка «Работы» (представление vCAPREP_LIST) и списка «Оценки соответствия критериям отбора» (представление vCAPREP_EVALUATION_LIST), а также сводного табличного отчёта «Капремонт» (представление CAPREP_LINE).

Процедура планирования программы капитального ремонта МД заключается в формировании списка необходимых работ для конкретных многоквартирных домов (Рисунок **36**). Для этого в представлении vR_CAPITAL_REPAIR указывается адрес МД, а в деталях «Работы» (представление vCAPREP_LIST) необходимые работы, их стоимость и площадь работ. Кроме работ в деталях «Оценки соответствия критериям отбора»

указываются наименование оценок и их значимость в виде баллов. В результате в представлении vR_CAPITAL_REPAIR автоматически рассчитываются необходимая сумма для ремонта и сумма баллов, что позволяет отбирать для программы капитального ремонта МД, ремонт в которых принесёт наибольший эффект.

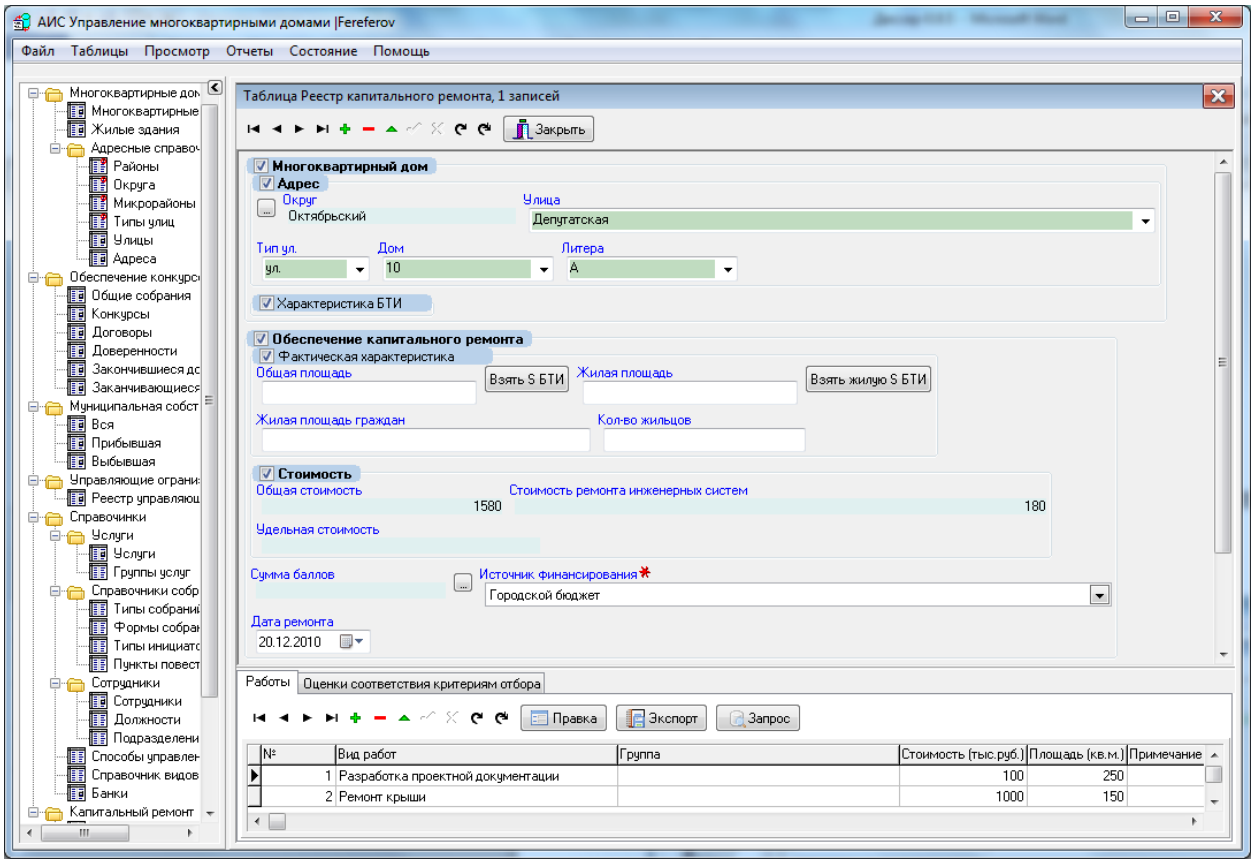

Рисунок 36. Реестр капитального ремонта

Представление CAPREP_LINE реализует отчёт согласно форме п.3 ст. 15 185-ФЗ, отправляемый в вышестоящие органы местного самоуправления.

Подсистема «Земельные участки МД» нацелена на решение задач связанных с пространственно-распределёнными объектами. Подсистема реализована с применением подсистемы «Карта» (Рисунок **37**). В подсистеме используется цифровая карта масштаба 1:2000 «Адресный план», содержащая информацию о зданиях и дорожной сети г. Иркутска. В подсистеме «Карта», кроме стандартных ГИС-функций, реализованы функция поиска зданий по адресу и привязка объектов карты с записями БД УМД.

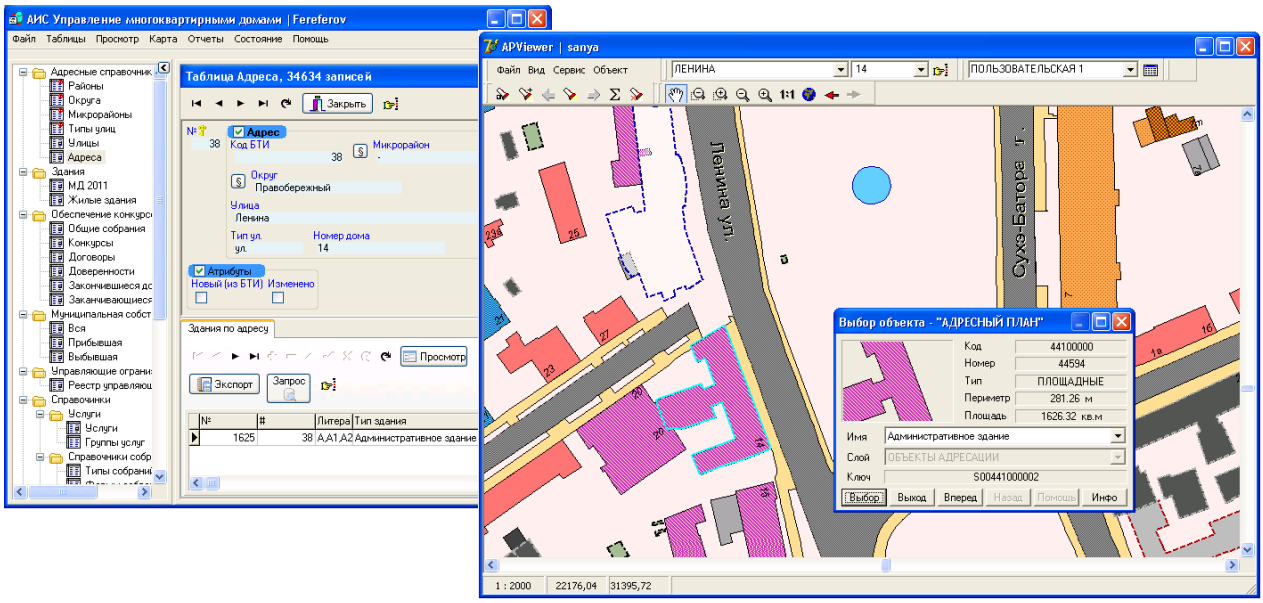

Рисунок 37. Взаимодействие с подсистемой «Карта»

Взаимодействие с цифровой картой города позволяет сотрудникам КЖКХ решать задачи выделения придомовых территорий МД, определение зон ответственности УО за уборку территорий, определение точек подвода коммуникаций ЖКУ к многоквартирным домам, а также решения ряда задач связанных с благоустройством территории города.

Подсистема «Расчёт платы нанимателей» предназначена для расчёта платы за пользование жилым помещением (плата за наем) для нанимателей жилых помещений по договорам социального найма и договорам найма жилых помещений государственного жилищного фонда, расположенного на территории г. Иркутска, и муниципального жилищного фонда г. Иркутска.

Расчёт платы выполняется по формуле утверждённой в «Решении думы г. Иркутска об установлении размера платы за жилое помещение» от 22 декабря 2005 г.:

$$П = Н_б * К_{зоны} * К_{кап} * К_{эт} * К_{бл} * S$$

где:

$Н_б$ - базовая ставка платы за пользование жилым помещением (платы за наем) для нанимателей жилых помещений по договорам социального найма и

договорам найма жилых помещений государственного жилищного фонда, расположенного на территории г. Иркутска, и муниципального жилищного фонда г. Иркутска, руб./мес. за 1 кв.м общей (жилой - для нанимателей, проживающих в коммунальных квартирах, общежитиях) площади. Базовые ставки утверждаются мэром каждый год и заносятся в АИС УМД сотрудниками КЖКХ в таблицу BaseRate.

$К_{зоны}$ - коэффициент месторасположения дома. На территории города выделены 11 зон, каждой из которых сопоставлен определённый коэффициент в зависимости от удалённости от центра города. Коэффициенты зон хранятся в таблице TerZone, связанной с адресами зданий.

$К_{кап}$ - коэффициент степени капитальности дома (материал стен). Выделены три степени капитальности: дома с кирпичными стенами, дома с крупнопанельными, блочными стенами, дома с деревянными стенами. Каждому зданию (записи из таблицы V_Здания) сопоставляется коэффициент степени капитальности через связь с таблицей WallMaterial.

$К_{эт}$ - коэффициент этажности. Для квартир расположенных на первых этажей задаётся понижающий коэффициент. Этаж квартир указан в представлении vKV, являющимся деталями для записей vUMD_LNK (Многоквартирные дома).

$К_{бл}$ - коэффициент степени благоустройства жилого помещения. Выделены три степени благоустройства: благоустроенное жилое помещение; жилое помещение, имеющее не все виды благоустройства; неблагоустроенное жилое помещение. Степени благоустройства хранятся в таблице ComfortDegree и сопоставлены со зданиями (связь с таблицей V_Здания).

S – площадь жилого помещения. Для квартир учитывается общая площадь, а для комнат в коммунальных квартирах и общежитиях – жилая. Показатели площади указаны в представлении vKV.

Подсистема реализована в виде приложения-надстройки (библиотека Payment.dll), вызываемого для списка МД (представление vUMD_LNK). Описание надстройки в спецификации:

```
PLUGIN PAYMENT PATH='Payment.dll' PROC=Init BROWSE
     BTN=(TBL=vUMD_LNK CAPTION='Плата нанимателей'
          HINT='Расчёт платы нанимателей')
```

Базовые ставки платы за пользование жилым помещением ($H_б$) приложение-надстройка получает непосредственно через SQL-запрос к справочнику BaseRate, используя параметры соединения из АИС УМД. Атрибуты $K_{зоны}$, $K_{кап}$, $K_{бл}$ включены из связанных справочников в состав полей представления vUMD_LNK как невидимые (не отображаемые пользователю). Приложение-надстройка, перебирая записи vUMD_LNK, получает эти коэффициенты для каждого МД, а также в цикле для каждого МД перебирает записи таблицы-деталей «Жилые помещения» (vKV) и, если помещение является муниципальным, получает $K_{эт}$, общую и жилую площади помещения ($S_{общ}$, $S_{жил}$) В зависимости от типа жилого помещения (квартира или комната) при расчёте платы применяются разные базовые ставки, начисляемые в одном случае на общую площадь, в другом на жилую. В общем виде выполнение расчёта платы за пользование жилыми помещениями представлено на схеме (Рисунок **38**):

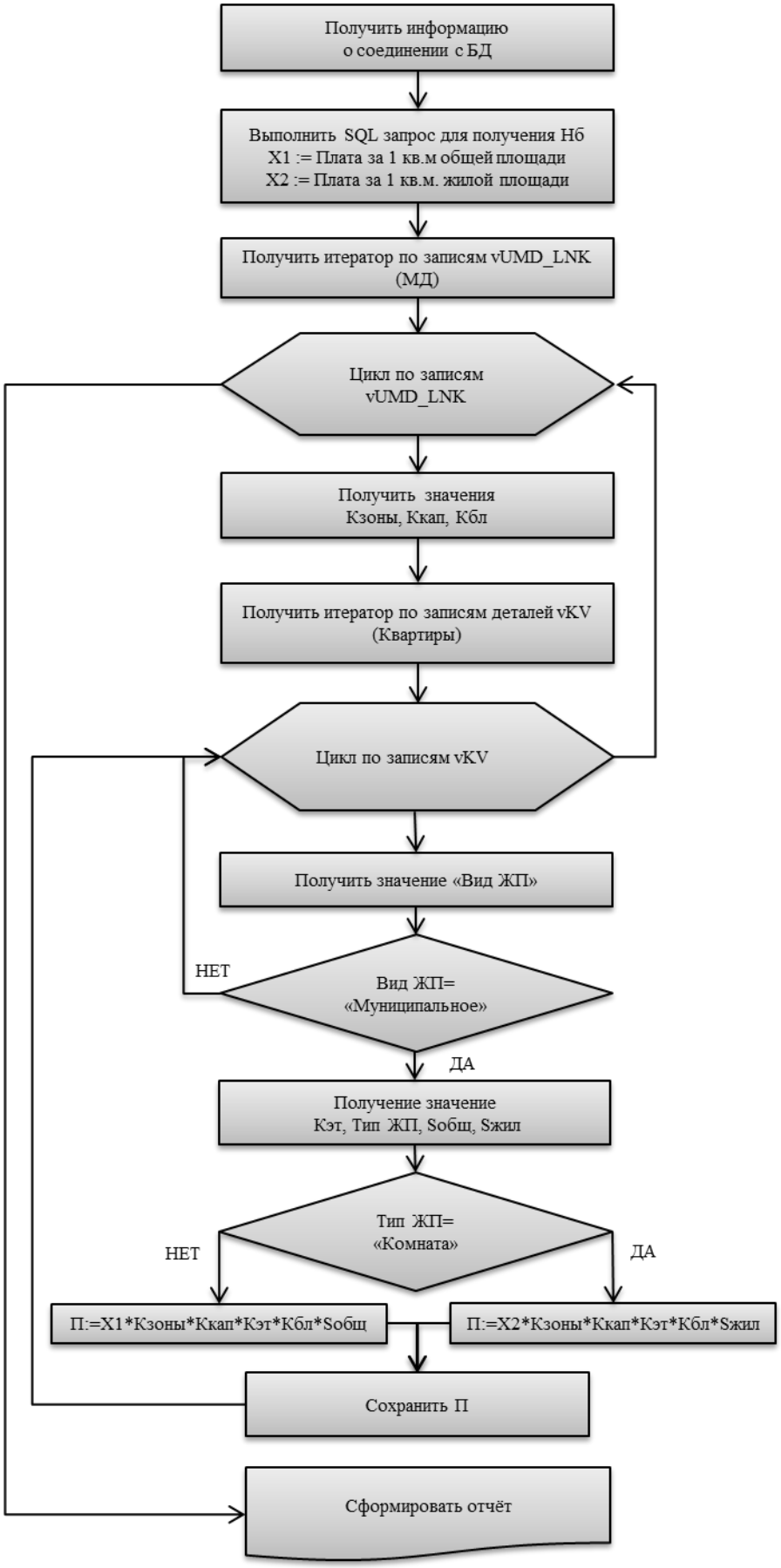

Рисунок 38. Схема расчёта платы за пользование жилыми помещениями

ЗАКЛЮЧЕНИЕ

В заключении отметим, что в диссертации представлен новый подход к автоматизации разработки ППС для работы с базами данных и обладающих ГИС-функциональностью, позволяющий сократить сроки и стоимость разработки за счёт применения оригинальной технологии описания структуры ПБД в виде декларативных спецификаций.

Результаты, выносимые на защиту:

1. Разработана технология автоматизации создания ПБД, отличием которой от известных является выделение информации о структуре ПБД, механизме взаимодействия с внешними ППС, а также с ГИС, и формирование спецификаций в виде формализованных знаний.

2. Создана оригинальная концептуальная модель ПБД, особенность которой заключается в том, что информация о структуре БД расширена знаниями о способах представления данных пользователю, а также механизме взаимодействия с внешними ППС и с ГИС.

3. Создан новый декларативный язык спецификаций ПБД, включающий конструкции для описания не только структур таблиц и связей между ними, но и правил формирования пользовательского интерфейса для взаимодействия с этими таблицами, взаимосвязи информации из БД с пространственными данными, а также механизма взаимодействия с внешними ППС, решающими специфические задачи.

4. Создано инструментальное средство, позволяющее интерактивно разрабатывать спецификации ПБД, обладающих ГИС-функциональностью и возможностью взаимодействия с внешними ППС, а также настраиваться при помощи спецификаций на работу с предметной БД.

Разработанное инструментальное средство ориентировано на разработку настольных ПБД работающих под управлением ОС Windows для взаимодействия с любыми реляционными СУБД, к которым существует возможность подключения через ADO. Предложенная технология имеет потенциал развития в

сторону web-приложений (в настоящий момент реализованы функции публикации БД, связанных с пространственнвми объектами). Применение инструментальной системы «ГеоАРМ» при создании ряда ППС [10, 11, 13, 15, 20] показало достаточную простоту и лёгкость использования и обеспечило значительное сокращение трудозатрат на этапах разработки и последующей модернизации ППС.

Список сокращений и условных обозначений

ADO -- ActiveX Data Objects

API – Application Programming Interface

BDE -- Borland Database Engine

EAV – EntityAttributeValue

IDE – Integrated Development Environment

MDA -- Model Driven Architecture

ORM – Object-Relational Mapping

UML -- Unified Modeling Language

VCL -- Visual Component Library

АИС -- Автоматизированная информационная система

АРМ -- Автоматизированное рабочее место

БД -- База данных

ППС – Прикладная программная система

АИС – Автоматизированная Информационная Система

ИС -- Информационная система

ИАС -- Информационно-аналитическая система

ПБД – Приложение баз данных

СУБД -- Система управления базами данных

ГИС -- Гоинформационная система

ЦК – Цифровая Карта

Литература

1. Агафонов В.Н. Спецификация программ: понятийные средства и их организация. Новосибирск: Наука, 1987. 240 с.
2. Агафонов В.Н. Требования и спецификации в разработке программ. М.: Мир, 1984.
3. Ахо А.В., Лам М.С. , Сети Р., Ульман Дж.Д.. Компиляторы: принципы, технологии и инструментарий. 2-е изд. М.: Вильямс, 2010. 1184 с.
4. Ахтырченко К.В., Сорокваша Т.П. Методы и технологии реинжиниринга ИС // Труды Института системного программирования. 2003. С. 116-128.
5. Библиотека классов платформы.NET Framework [Электронный ресурс]. URL: http://msdn.microsoft.com/ru-ru/library/gg145045(v=vs.110).aspx (дата обращения: 12.09.2013).
6. Буч Г., Максимчук Р.А. , Энгл М.У., Янг Б.Дж., Коналлен Д., Хьюстон К.А. Объектно-ориентированный анализ и проектирование с примерами приложений. Вильямс, 2010. 720 с.
7. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. 2-е изд. СПб.: ДМК Пресс, 2004. 432 с.
8. Бычков И. В., Гаченко А. С., Хмельнов А. Е., Фереферов Е. С. Система создания автоматизированных рабочих мест с возможностью взаимодействия с пространственными данными на основе метаописаний структур баз данных // Современные технологии. Системный анализ. Моделирование. 2008. Спец. вып. С. 12-17.
9. Бычков И.В., Васильев С.Н., Опарин Г.А., Ружников Г.М., Шелехов В.А. Становление и развитие информационно-вычислительных технологий и инфраструктуры в Иркутске под руководством В.М. Матросова, их современное состояние и перспективы // Аналитическая механика, устойчивость и управление: Тр. X Междунар. Четаевской конф. Казань, 2012. Т. 4. С. 29-46.

10. Бычков И.В., Гаченко А.С., Попова А.К., Ружников Г.М., Фереферов Е.С., Хмельнов А.Е. Применение ГИС- и Веб- технологий для создания интегрированных информационно-аналитических систем // Вычислительные технологии. 2007. Т. 12, Спец. вып. 3. С. 5-18.
11. Бычков И.В., Гаченко А.С., Ружников Г.М., Маджара Т.И., Фереферов Е.С., Хмельнов А.Е. Внедрение современных информационных технологий в региональных проектах // Вестник НГУ. 2008. Т. 6. № 1. С. 15-24.
12. Бычков И.В., Гаченко А.С., Фереферов Е.С., Хмельнов А.Е. Программный комплекс для создания АРМ с картографической привязкой с использованием метаописаний структуры баз данных (ГеоАРМ): Свидетельство о государственной регистрации программ для ЭВМ № 2007613643 от 27.08.2007. М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам, 2007.
13. Бычков И.В., Ружников Г.М., Хмельнов А.Е., Фёдоров Р.К., Гаченко А.С., Фереферов Е.С., Шигаров А.О. Программная система актуализации векторной карты зданий и сооружений по космоснимку // Горный информ.-аналит. бюл. (науч.-техн. журн.). 2009. № ОВ 18 С. 141-145.
14. Бычков И.В., Ружников Г.М., Хмельнов А.Е., Фереферов Е.С. Библиотека для создания отчетов с использованием метаописаний структур БД и шаблонов, содержащих метки форматирования данных: Свидетельство о государственной регистрации программ для ЭВМ № 2011618280 от 20.10.2011. М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам, 2011.
15. Бычков И.В., Ружников Г.М., Хмельнов А.Е., Шигаров А.О., Гаченко А.С., Фёдоров Р.К., Фереферов Е.С., Попова А.К., Новицкий Ю.А. Монография. Интеграция информационно-аналитических ресурсов и обработка пространственных данных в задачах управления территориальным развитием. Новосибирск: Изд-во СО РАН, 2011. 369 с.
16. Бычков И.В., Ружников Г.М., Хмельнов.А.Е, Фереферов Е.С. Программный

модуль, реализующий интерфейс пользователя ГИС Адресный план (Интерфейс АП)). Свидетельство о государственной регистрации программ для ЭВМ № 2010612640 от 16.04.2010. М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам., 2010.

17. Вендров А.М. Case-технологии. М.: Финансы и статистика, 1998. 176 с.
18. Воровский Ф.С. Информатика. Новый систематизированный толковый словарь-справочник. 3-е изд. М.: Физматлит, 2003. 760 с.
19. Гантер Р. Методы управления проектированием программного обеспечения. М. : Мир, 1981. 392 с.
20. Гаченко А. С., Ружников Г. М., Фереферов Е. С., Хмельнов А. Е. Разработка информационной системы обеспечения градостроительной деятельности в муниципальных образованиях // Вестник НГУ. 2008. Т. 6. № 3. С. 72-79.
21. ГИС ассоциация [Электронный ресурс]. URL: http://gisa.ru (дата обращения: 1.12.2013).
22. Грибачёв К. Delphi и Model Driven Architecture. Разработка приложений баз данных. СПб.: Питер, 2004.
23. Грибова В.В., Кисленок Р.С. Автоматизация разработки визуального представления пользовательского интерфейса по модели предметной области // Искусственный интеллект. 2006. № 4. С. 148-152.
24. Грибова В.В., Клещёв А.С. Онтологическая парадигма программирования // II Междунар. науч.-техн. конф. «Open Semantic Technologies for Intelligent Systems». Минск, 2012. С. 213-220.
25. Грибова В.В., Клещев А.С. Концепция разработки пользовательского интерфейса на основе онтологий // Вестник ДВО РАН. 2005. № 6. С. 123-128.
26. Дейт К. Дж. Введение в системы баз данных. 8-е изд. М.: Вильямс, 2005. 1328 с.
27. Дехтяренко И.А. Декларативное программирование // SoftCraft. 2003. URL: http://www.softcraft.ru/paradigm/dp/dp01.shtml (дата обращения: 14.09.2013).

28. Ершов А.П. Введение в теоретическое программирование (беседы о методе). М.: Наука, 1977. 288 с.
29. Культин Н. Основы программирования в Delphi XE. СПб.: БХВ-Петербург, 2011. 416 с.
30. Лавров С., Левин Д., Матулис В., Мацкин М., Нариньяни А., Опарин Г., Тыугу Э., Хорошевский В. Технологические системы поддержки разработок искусственного интеллекта // Представление знаний в человеко-машинных и робототехнических системах: отчет Проблемной комиссии многостороннего сотрудничества социалистических стран «Научные вопросы вычислительной техники». М.: Изд-во ВЦ АН СССР, ВИНИТИ, 1984. 102-123 с.
31. Лапшин В.А. Онтологии в компьютерных системах. М.: Научный мир., 2010. 224 с.
32. Макки А. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов. М.: Вильямс, 2010. 416 с.
33. Массель Л.В., Копайгородский А.Н., Аршинский В.Л. Построение интеллектуальных систем для исследований энергетики на основе алгебраических сетей и онтологии: подход и реализация и реализация // Вычислительные технологии. 2008. Т. 13. № S1. С. 50-58.
34. Монахов В.В. Язык программирования Java и среда NetBeans. 3-е изд. СПб.: BHV, 2011. 704 с.
35. Одинцов И.О. Профессиональное программирование. Системный подход. 2-е изд. СПб.: БХВ-Петербург, 2004. 624 с.
36. Опарин Г. А. Сатурн – метасистема для построения пакетов прикладных программ // Пакеты прикладных программ. Методы и разработки. Новосибирск: Наука, 1982. С. 130-160.
37. Осипов Д. Базы данных и Delphi. Теория и практика. СПб.: БХВ-Петербург, 2011. 752 с.
38. Павлов А.И. Инструментальное средство для создания гибких

информационно-аналитических систем: дис. … канд. техн. наук: специальность 05.13.11., Иркутск. 2005.

39. Палмер С.Р., Фелсинг Д.М. Практическое руководство по функционально-ориентированной разработке ПО. М.: Вильямс, 2002. 299 с.
40. Сафронов И. Visual Basic в задачах и примерах. СПб.: БХВ-Петербург, 2008. 400 с.
41. Склонение фамилий, имен и отчеств по падежам Библиотека функций. [Электронный ресурс] // Королевство Delphi. Виртуальный клуб программистов.: [сайт]. URL: http://www.delphikingdom.com/asp/viewitem.asp?catalogid=412 (дата обращения: 23.01.2014).
42. Средство разработки ГИС-приложений GIS ToolKit [Электронный ресурс]. URL: http://gisinfo.ru/products/gistool_win.htm (дата обращения: 17.01.2014).
43. Тихомиров Ю. Самоучитель MFC. Книга по Требованию, 2012.
44. Тыугу Э.Х. Концептуальное программирование. М.: Наука, 1984. 256 с.
45. Фаронов В.В. Программирование баз данных в Delphi 7. СПб.: Питер, 2004. 459 с.
46. Фёдоров Р.К., Бычков И.В., Хмельнов А.Е., Новицкий Ю.А., Ружников Г.М., Гаченко А.С., Фереферов Е.С., Шигаров А.О., Парамонов В.В., Попова А.К. Разработка геоинформационной системы "Адресный план" г. Иркутска // Современный технологии. Системный анализ. Моделирование. 2009. № 3(23). С. 14-19.
47. Фереферов Е. С., Бычков И. В., Хмельнов А. Е. Технология создания автоматизированных рабочих мест с возможностью обработки пространственных данных на основе метаописаний структур баз данных // Вестник ИрГТУ. 2006. Т. 3. № 2(26). С. 52-59.
48. Фереферов Е. С., Новицкий Ю. А., Ружников Г. М., Хмельнов А. Е. Технология интеграции геоинформационных функций в информационные системы // Вестник Бурятского гос. ун-та. 2012. № 9. С. 59-63.

49. Фереферов Е. С., Хмельнов А. Е. Модель информационной системы для автоматизации создания приложений баз данных с ГИС-функциональностью. // Тр. XVIII Байкальской всерос. конф. «Информационные и математические технологии в науке и управлении». Иркутск. 2013. С. 200-207.
50. Фереферов Е.С., Бычков И.В., Новицкий Ю.А., Ружников Г.М., Хмельнов А.Е. Организация работы с электронными картами исключающая утечку векторной информации // Горный информ.-аналит. бюл. (науч.-техн. журн.). 2009. № ОВ 18. С. 220-224.
51. Фереферов Е.С., Бычков И.В., Ружников Г.М., Хмельнов А.Е. Инструментальное средство автоматизации создания приложений баз данных на основе декларативных спецификаций // Вестник Бурятского гос. ун-та. 2011. № 9. С. 118-122.
52. Фереферов Е.С., Бычков И.В., Ружников Г.М., Хмельнов А.Е. Технология интеграции баз данных на основе декларативных спецификаций // Справочник конференции «Математические и информационные технологии». Белград, 2011. С. 76.
53. Фереферов Е.С., Бычков И.В., Ружников Г.М., Хмельнов А.Е. Технология создания автоматизированных информационных систем на основе декларативных моделей. // Сборник тез. и докл. 17-й междунар. науч. конф. «Системный анализ, управление и навигация». М., 2012. С. 65-67.
54. Фереферов Е.С., Бычков И.В., Хмельнов А.Е. Метаописание баз данных как основа интеграции информационно-справочных систем и ГИС. // Вычислительные технологии. 2007. Т. 12. № 5. С. 41-51.
55. Фереферов Е.С., Бычков И.В., Хмельнов А.Е. Технология разработки приложений баз данных на основе декларативных спецификаций // Вычислительные технологии. 2014. Т. 19. № 5. С. 85-100.
56. Фереферов Е.С., Гаченко А.С., Ружников Г.М., Хмельнов А.Е. Муниципальная информационная система обеспечения градостроительной деятельности //

Вычислительные технологии. 2008. Т. 13, спец. вып. 1. С. 11-16.

57. Фереферов Е.С., Хмельнов А.Е. Автоматизация создания пользовательского интерфейса на основе модели приложения баз данных // Вестник Бурятского гос. ун-та. 2013. № 9. С. 100-118.
58. Фереферов Е.С., Хмельнов А.Е. Автоматизация создания пользовательского интерфейса на основе модели приложения баз данных // Тез. II Российско-монгольской конференции молодых ученых по математическому моделированию, вычислительно-информационным технологиям и управлению (г. Иркутск - п. Ханх). Иркутск. 2013. С. 63.
59. Фереферов Е.С., Хмельнов А.Е. Реализация менеджера размещения визуальных компонентов в Delphi // Материалы международной конференции «Вычислительные и информационные технологии в науке, технике и образовании». Алмааты-Новосибирск, 2008. Т. 13. С. 283-287.
60. Фереферов Е.С., Хмельнов А.Е. Язык представления баз данных // Материалы III Междунар. конф. «Инфокоммуникационные и вычислительные технологии и системы». Улан-Удэ. 2010. С. 269-272.
61. Хантер Р. Основные концепции компиляторов. М.: Вильямс, 2002. 256 с.
62. Хмельнов А.Е., Ружников Г.М., Фереферов Е.С. Построение сложных пользовательских запросов с использованием спецификаций структуры БД. // Материалы III Междунар. конф. «Инфокоммуникационные и вычислительные технологии и системы». Улан-Удэ, 2010. С. 275-279.
63. Цейтин Г.С. На пути к сборочному программированию // Програмирование. 1990. № 1. С. 78-92.
64. Чарнецки К., Айзенекер У. Порождающее программирование. Методы, инструменты, применение. Для профессионалов. СПб.: Питер, 2005. 736 с.
65. Черкашин Е.А., Федоров Р.К., Бычков И.В., Парамонов В.В. Автоматизация синтеза ядра информационной системы с использованием UML-описания // Вычислительные технологии. 2005. Т. 10. С. 114-121.

66. Черткова Е.А. Применение модельных каркасов для разработки графического пользовательского интерфейса. // Вестник Астраханского гос. техн. ун-та. 2007. № 1(36). С. 150-153.
67. Шигаров А. О., Парамонов В. В., Попова А. К., Бычков И. В., Ружников Г. М., Хмельнов А. Е., Новицкий Ю. А., Фёдоров Р. К., Гаченко А. С., Фереферов Е. С. Технология создания и ведения информационной системы «Адресный план» с использованием крупномасштабных электронных карт // Горный информ.-аналит. бюл. (науч.-техн. журн.). 2009. № ОВ 18. С. 176-180.
68. Элиенс А. Принципы объектно-ориентированной разработки программ. 2-е изд. Вильямс, 2002. 496 с.
69. Янкевич А.А. Графическая модель для спецификации и синтеза интерфейса пользователя автоматизированных систем: дис. … канд. техн. наук: специальность 05.13.11. С-Пб., 2004.
70. ADO Programmer's Guide [Электронный ресурс]. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ms681025(v=vs.85).aspx (дата обращения: 08.11.2013).
71. Ambler S. Mapping Objects to Relational Databases: O/R Mapping In Detail // Agile Data Home Page. 2013. URL: http://www.agiledata.org/essays/mappingObjects.html (дата обращения: 16.05.2014).
72. ArcView [Электронный ресурс]. URL: http://www.esri.com/software/arcgis/arcview (дата обращения: 12.10.2013).
73. Burbeck S., Ph.D. // Applications Programming in Smalltalk-80™: How to use Model-View-Controller (MVC). URL: http://www.math.sfedu.ru/smalltalk/gui/mvc.pdf (дата обращения: 23.08.2013).
74. Berstel J., Reghizzi S., Roussel G., San Pierto P. A Scalable Formal Method for Design and Automatic Checking of User Interfaces // ACM Transactions on Software Engineering and Methodolodgy. 2005. Vol. 14. No. 2. P. 124-167.
75. Bohm C., Jacopini G. Flow Diagrams, Turing Machines and Languages with Only

Two Formation Rules // Communications of the ACM. 1966. Vol. 9. No. 5. P. 366-371.

76. Class FlowLayout [Электронный ресурс] // Java™ Platform Standart Ed.7: [сайт]. URL: http://docs.oracle.com/javase/7/docs/api/java/awt/FlowLayout.html (дата обращения: 05.07.2013).
77. Hibernate ORM documentation [Электронный ресурс] // Hibernate ORM: [сайт]. URL: http://hibernate.org/orm/documentation/ (дата обращения: 13.02.2014).
78. IBM – Rational Rose Enterprise [Электронный ресурс]. URL: http://www-03.ibm.com/software/products/ru/enterprise/ (дата обращения: 12.01.2014).
79. Lerman J. Programming Entity Framework: Building Data Centric Apps with the ADO.NET Entity Framework. 2nd ed. O'Reilly Media, 2010. 920 p.
80. Lloyd J.W. Joint Conference on Declarative Programming, GULP-PRODE. // Practical advantages of declarative programming. 1994. P. 94.
81. Maguire St. Writing Solid Code. Microsoft Press, 1993. 256 p.
82. MapInfo [Электронный ресурс]. URL: http://www.mapinfo.com/ (дата обращения: 20.09.2013).
83. Martin J. RAD, Rapid Application Development. New York: MacMillan Publishing Co., 1991. 788 p.
84. MicroStation: среда информационного моделирования [Электронный ресурс]. URL: http://www.bentley.com/ru-RU/Products/microstation+product+line/ (дата обращения: 15.10.2013).
85. OMG Model Driven Architecture [Электронный ресурс]. URL: http:// www.omg.org/mda/ (дата обращения: 10.1.2014).
86. Pubs Sample Database [Электронный ресурс] // Microsoft SQL Server: [сайт]. URL: http://technet.microsoft.com/en-us/library/aa238305(v=sql.80).aspx (дата обращения: 12.05.2013).
87. Silva P.P., Griffiths T., Paton N.W. International Working Conf. on Advance Visual Interfaces 2000 (AVI2000) // Generating User Interface Code in a Model Based

User Interface Development Environment. Palermo (Italy). 2000. P. 155-160.

88. Sybase CIS PowerDesigner [Электронный ресурс]. URL: http://www.sybase.ru/products/powerdesigner (дата обращения: 12.01.2014).
89. Using Windows Script Files (.wsf) [Электронный ресурс] // Microsoft Developer Network: [сайт]. URL: http://msdn.microsoft.com/en-us/library/15x4407c.aspx (дата обращения: 17.06.2013).
90. VCL Overview - RAD Studio [Электронный ресурс]. URL: http://docwiki.embarcadero.com/RADStudio/XE5/en/VCL_Overview (дата обращения: 15.08.2013).
91. W3C HTML [Электронный ресурс]. URL: http://www.w3.org/html/ (дата обращения: 9.12.2013).
92. Weisfeld M. The Object-Oriented Thought Process. 3rd ed. Addison-Wesley Professional, 2008. 360 p.

Приложение А. Семантика языка спецификаций ПБД

Для описания конструкций синтаксиса ЯПБД будем использовать нотацию, аналогичную EBNF (Extended Backus-Naur Format - Расширенный формат Бэкуса-Наура). Символы этой нотации имеют следующий смысл:

`:=` – является по определению

`|` – или

`< >` – нетерминальный символ, представляемый заключенным в скобки понятием

`"текст"` – литерал

`*` – возможность повторения предшествующей синтаксической конструкции нуль или более раз

`+` – возможность повторения предшествующей синтаксической конструкции один или более раз

`{ }` – заключенные в скобки синтаксические конструкции рассматриваются как единая конструкция

`[ ]` – заключенная в скобки синтаксическая конструкция является необязательной.

Язык представления баз данных принадлежит к классу LL(1) […] грамматик. Предложения, написанные на ЯПБД, имеют следующий вид:

`<Предложение ЯПБД> := <Стартовое слово> <Список обязательных выражений> [список необязательных выражений]`

`<Стартовое слово> := “ADO”| “BDE”| “CFG”| “PLUGINS”| “TBL”| “DSPL”| “MENU”| “MAP”`

`<Список обязательных выражений>` и `[список необязательных выражений]` зависят от стартового слова.

Описание подключения к базе данных.

В ЯПБД поддерживается описание соединения с БД в двух технологиях - BDE и ADO.

<Строка соединения в технологии BDE>:= “BDE” <Параметры соединения>

<Параметры соединения>:= “ALIAS”“=”<Псевдоним БД> “USER”“=”<Имя пользователя> “PASSWORD”“=”<Пароль>

<Псевдоним БД>:=<Буква>*|<Цифра>*

<Имя пользователя>:=<Буква>*|<Цифра>*

<Пароль>:=<Буква>*|<Цифра>*

<Буква> := A|B|C|…|Z

<Цифра> := 0|1| … |9

<Строка соединения в технологии ADO>:= “ADO” <Строка соединения> [<Параметры соединения>]

<Строка соединения>:= “CONNECTIONSTRING” “=”<Строка соединения ADO> (вид строки соединения согласно MSDN[])

<Параметры соединения>:= “LOGINPROMT” “CONNECTIONTIMEOUT”“=”<Цифра>* “COMMANDTIMEOUT” “=”<Цифра>* “ServerCursors”

Присутствие параметра LOGINPROMT указывает на необходимость запроса имени пользователя и пароля при подключении к БД. CONNECTIONTIMEOUT и COMMANDTIMEOUT позволяют настроить время ожидания при подключении к БД и выполнении команд запросов. ServerCursors указывает на необходимость использования серверных курсоров.

Описание общих настроек системы.

<Общие настройки системы>:= “CFG” <Параметры настройки>

<Параметры настройки>:= “Qalways” “LUPKFMT”“=”<Стр. фмт.> “luNameFmt”“=”<Стр. фмт.> “QuoteFNFmt”“=”<Стр. фмт.> “QuoteQFNFmt”“=”<Стр. фмт.> “QuoteTNFmt”“=”<Стр. фмт.> “UpdateMode”“=”0|1|2 “SCHEMA”“=”<Схема> “DATEFMT”“=”<Дата фмт.> “LogSQL” “LogLU” “FIXED” “AUTOINCPK” “APPNAME”“=”<Имя приложения>

“APPTITLE”“=”<Заголовок приложения> “GUISTYLE”“=”<Цифра> “TOPREFS” “AUTOINDEX” “PreviewDefaults” “DBCaseIns” “SplitScreen”“=”“V”|“H”

<Стр. фмт.>:= <Символ>* “%s” <Символ>*

<Символ>:= <Буква>| “.” | “[” | “]”

<Схема>:=“.”<Буква>*

%s – маска строки форматирования передаваемая в функцию Format.

Назначение параметров:

Qalways – параметр означает, что при работе с БД необходимо всегда использовать запросы (TQuery);

LUPKFMT – строка форматирования для получения наименования поля первичного ключа справочника по имени таблицы справочника;

LuNameFmt – строка форматирования для получения наименования поля наименования справочника по имени таблицы справочника;

QuoteFNFmt – строка форматирования, задающая кавычки для включения в запрос поля с недопустимыми символами в имени.

QuoteQFNFmt – строка форматирования, задающая кавычки для включения в запрос псевдонима для столбца результата;

QuoteTNFmt – строка форматирования, задающая кавычки для включения в запрос имени таблицы;

UpdateMode – параметр, позволяющий выбрать способ обновления записей (используется только при работе через BDE). Значения: 0-upWhereAll, 1-upWhereChanged, 2-upWhereKeyOnly (используется по умолчанию);

SCHEMA – наименование схемы, в которой находится таблица, например, “dbo” или “AnotherSQL.AnotherBase.dbo”. Используется для тех таблиц, у которых не задано при описании наименование таблицы в БД. Наименование таблицы в БД с использованием SCHEMA имеет вид:

SCHEMA “.” <Наименование таблицы>;

DATEFMT – Формат даты для включения констант типа дата в запрос.

LogSQL – параметр, включающий режим записи в журнал текстов сгенерированных запросов (используется при отладке);

LogLU – параметр, включающий режим записи в журнал сообщений о создании справочников (используется при отладке);

FIXED – параметр обозначающий, что спецификация является окончательной и не требует дальнейшего редактирования. При включении этого флага в приложении блокируются все функции администрирования;

AUTOINCPK - параметр означает, что все первичные ключи таблиц, состоящие из одного поля, являются автоинкрементными полями;

APPNAME – Наименование приложения (отображается в заголовке главного окна);

APPTITLE – Заголовок приложения (отображается на панели задач и при переключении по Alt+Tab). Если не указан, то принимает значения 'ГеоАРМ' или 'Редактор БД' (в зависимости от варианта приложения);

GUISTYLE – стиль пользовательского интерфейса. Может принимать значения: 0 – все формы открываются в отдельных окнах, дерево таблиц отсутствует, имеется только меню для выбора таблицы; 1 – все формы открываются в одном главном окне, в котором отображается дерево таблиц и текущая форма;

TOPREFS – при установке этого флага на формах приложения отображаются только кнопки для перехода к таблицам, доступным через ссылки верхнего уровня. Для таблиц, доступных по более глубоким ссылкам (через несколько уровней) кнопки при этом не создаются;

AUTOINDEX – данный параметр позволяет при работе через BDE с локальными таблицами (Paradox, DBF) автоматически создавать недостающие (для работы согласно имеющимся описаниям) индексы;

PreviewDefaults - при указании этого флага после добавления записи сразу (до её сохранения) отображаются значения полей по умолчанию. В том случае, когда поле имеет значение по умолчанию, заданное выражением или функцией, для получения каждого такого значения выполняется отдельный запрос к серверу.

Включение режима PreviewDefaults позволяет заранее информировать пользователя о том, что поле получит значение, даже если он это поле не заполнит;

DBCaseIns – установка этого флага означает, что в базе данных не различаются строчные и прописные буквы, и поэтому, например, не требуется использовать функцию UPPER для сравнения строк без учёта регистра;

SplitScreen – параметр позволяет задать заранее взаимное расположение окна для работы с БД и Картой. Значение “H” соответствует команде «Разделить горизонтально», “V” – «Разделить вертикально».

Описание структур таблиц.

<Описание таблицы>:= “TABLE” <Псевдоним таблицы> [“FOR” <Наименование таблицы в БД>] [“AS” <Отображаемое имя таблицы>] [<Свойства таблицы>]

“FIELDS” “(” <Описание свойств и вида полей> “)”

[“REFS”(<Описание связей>)]

<Псевдоним таблицы>:=<Буква>*|<Цифра>* – задаётся для идентификации таблицы в системе. В большинстве случаев совпадает с именем реальной таблицы БД.

<Наименование таблицы в БД> := [<Схема>] <Буква>*| <Цифра>* – задаётся в случае, если <Псевдоним таблицы> отличается от имени реальной таблицы БД.

<Отображаемое имя таблицы>:=<Буква>*|<Цифра>* – имя, которое будет отображаться пользователю в заголовках форм при работе с этой таблицей.

<Свойства таблицы>:=“READONLY” “ORDER”“=”<Список полей> “NAMES”“=”<Список полей> “FILTER”“=”<Список полей> “MAPKIND”“=”<Цифра>* “MAPFLD”“=”<Имя поля> <Ограничение целостности> <Координаты объекта>

<Список полей>:= <Имя поля> [“,”<Имя поля>]*

<Имя поля>:= {<Буква>*|<Цифра>*}

<Ограничение целостности>:= “CHECK”“(” <Список полей> <Уровень реакция>“)”

<Уровень реакция>:= “abort” | “warn”

Назначение свойств таблицы:

READONLY – параметр задающий режим работы с таблицей «только чтение»

FILTER – задаёт условие для отбора записей из таблицы. Условие должно соответствовать правилам формирования значения для свойства Filter у TDataSet, при этом все наименования полей необходимо заключить в квадратные скобки (“[”,“]”).

NAMES – задаёт набор полей, именующих запись. Например, для таблицы адресов домов именующими полями могут быть «Улица», «Тип улицы», «Номер дома». Именующие поля могут соответствовать естественному первичному ключу записи (в отличие от искусственного автоинкрементного первичного ключа). Значения этих полей отображаются для пользователя при поиске записей, таблица сортируется в лексикографическом порядке по значениям именующих полей (если другой порядок не задан при помощи ключа ORDER).

ORDER –Задаёт набор полей, по которому сортируется таблица.

MAPKIND – код таблицы в таблице связи с картой. Используется для привязки записей таблицы к объектам карты.

MAPFLD – поле таблицы, по которому осуществляется связь записей с объектами карты. Поле должно быть целочисленным. Если имеется ключ MAPKIND, но отсутствует ключ MAPFLD, то в качестве такого поля используется поле первичного ключа (если оно единственное и целочисленное).

<Координаты объекта>:=“COORDS” “=”<Вид координат> “:”<Параметры> - позволяет описывать способ автоматического создания объекта на цифровой топооснове по координатам.

<Вид координат>:= “Point” | “Line” | “PolyLine” | “Polygon” | “PolyPolygon”

<Параметры>:=<Имя связи с подчинённой таблицей> “.” <Список полей>

Описание полей таблиц.

<Описание свойств и вида полей>:=(<Имя поля>“=”<Вид поля> [“AS” <Отображаемое имя>] [<Свойства поля>] “,”)*

<Вид поля>:= “P”|“I”|“F”|“B”|“D”|“G”|“X”|“S”|“N”|“L”|<Связь>

Значения видов полей:

“P” – поле, входит в первичный ключ таблицы. Такие поля должны перечисляться первыми в данной секции. Если не указывается тип поля при помощи других ключей, то поле считается целочисленным;

“I” – целочисленное поле;

“F” – числовое поле;

“B” – логическое поле;

“D” – поле даты;

“G” – поле с графическим изображением;

“X” – BLOB-поле, содержащее произвольный файл;

“S” – строковое поле. При формировании условий запроса пользователь будет иметь возможность задать условия на значение строки или её подстрок;

“N” – поле имени, т.е. строковое поле, уникально идентифицирующее запись данной таблицы. При формировании условий запроса пользователь будет иметь возможность выбрать интересующие его значения такого поля из списка;

“L” – списочное поле, т.е. строковое поле, которое может принимать ограниченное число значений (например, «да», «нет»). Необходимость использования таких полей возникает при описании БД, в которых разработчик поленился организовать справочник для хранения списка возможных значений. При формировании условий запроса пользователь, тем не менее, будет иметь возможность выбрать интересующие его значения такого поля из списка;

<Связь> := "^"<Наименование таблицы>["("<Имя поля>")"] – позволяет описать простую ссылку на таблицу или представление. Если ссылка указывает на поля первичного ключа целевой таблицы <Наименование таблицы>, то можно не указывать ["("<Имя поля>")"]. Если явно не указан иной тип поля, то ссылочное поле является целочисленным. Выражение данного вида кроме самого поля определяет связь с таблицей <Наименование таблицы>. При этом наименование создаваемой связи совпадает с наименованием поля.

<Отображаемое имя>:={<Буква>*|<Цифра>*}

<Свойства поля>:= "READONLY" <Ширина поля>

<Ширина поля>:="WIDTH""="<Цифра>+

Значение свойств полей:

READONLY – поле только для чтения.

WIDTH – задаёт видимую ширину поля в режиме табличного просмотра

Описание связей.

<Описание связей> := <Имя связи> "="{ {<Имя таблицы> "." <Имя поля>} | {<Имя поля> "^" <Имя таблицы>["("<Список полей>")"]"," }*

<Имя связи> := {<Буква>*|<Цифра>*}

Конструкция <Имя связи>"="<Имя таблицы>"."<Имя поля> позволяет описывать связи вида DR, т.е ссылку на таблицу «деталей» при наличии связи «Мастер-детали». При этом необходимо, чтобы в описании таблицы являющейся «деталями» была описана связь вида LR на таблицу «мастер».

Конструкция <Имя связи>"="<Имя поля>"^"<Имя таблицы> позволяет описывать дополнительные связи из поля <Имя поля> в ключевые поля таблицы <Имя таблицы> или в указанные поля. Указание таких связей бывает необходимо в случае нескольких связей из одного поля или связей по нескольким полям.

Описание представлений.

<Представление>:= “VIEW” <Имя представления> “FOR” <Псевдоним базовой таблицы>

[“AS” <Отображаемое имя представления>]

[<Свойства представления>]

“FIELDS” “(“ <Описание полей представления> “)”.

<Имя представления>:={<Буква>*|<Цифра>*}.

<Псевдоним базовой таблицы>:={<Буква>*|<Цифра>*} – базовая таблица должна быть описана в спецификации.

<Отображаемое имя представления>:={<Буква>*|<Цифра>*}.

<Свойства представления>:=<Свойства таблицы> - свойства представлений аналогичны свойствам таблиц (описаны на стр.46)

<Описание полей представления>:={<Описание поля>“,”}*

<Описание поля>:= <Имя поля>“=”[{<Имя связи>“.”}*<Имя поля>] [“AS” <Отображаемое имя>]

В описании полей представления в качестве атрибутов указываются имена полей, а в качестве значений атрибутов - путь получения значения поля представления из полей базовой таблицы или из таблиц, на которые имеются ссылки из базовой таблицы. Если путь пустой, то поле берётся из одноимённого поля базовой таблицы представления. Если путь имеет вид <Имя связи>“.”<Имя поля>, то поле берётся из поля <Имя поля> таблицы (представления), на которую указывает последовательность ссылок <Имя связи>.

Описание форм.

Описание экранных форм является не обязательным, поскольку система «ГеоАРМ» после интерпретации спецификации автоматически формирует визуальный интерфейс для работы с таблицами и представлениями. В частности для работы с записями автоматически создаются пользовательские формы. При этом применяется алгоритм динамической компоновки, позволяющий в зависимости от размера экрана наиболее эффективно размещать элементы управления на форме. Иногда требуется объединять поля в смысловые группы и задавать их порядок, что улучшает восприятие информации и повышает удобство

работы пользователей. В ЯПБД предусмотрены следующие конструкции для управления формированием пользовательских форм, которые указываются непосредственно при описании полей таблиц или представлений.

<Элемент группировки>:= "[" <Вид группы> "=" <Наименование группы> <Список полей> "]"

<Вид группы>:= "GB" | "TS" | "DTS" | "DGB"

"GB" предписывает объединить поля в блок обрамлённый рамкой (GroupBox). "TS" позволяет выделить поля на отдельную закладку (TabSheet). "DGB" указывается для вставки блока с таблицей «деталей» в нужное место формы, при этом <Наименование группы> должно совпадать с именем связи «деталей», а <Список полей> отсутствует. "DTS" позволяет вставить закладку с таблицей деталей внизу формы.

Блоки типа "GB" и "DGB" могут быть вложены в другие блоки "GB" или в закладки "TS".

Описание надстроек.

<Надстройка> := "PLUGIN" <Имя надстройки> <Параметры надстройки>

<Имя надстройки> := {<Буква>*|<Цифра>*}.

<Параметры надстройки> := <Путь к файлу надстройке> <Имя точки входа> <Режим> <Обновить> <Описание кнопок вызова надстроек>

<Путь к файлу надстройке>:= "PATH" "="<Путь> – задаёт путь файла динамической библиотеки, содержащей реализацию модуля расширения. Пути задаются или абсолютно, или относительно папки Plugins в каталоге исполняемого файла;

<Имя точки входа>:= "PROC" "="<Имя>– задаёт имя точки входа в динамической библиотеке, содержащей реализацию модуля расширения (т.е. в одной DLL можно реализовать несколько модулей расширения).

<Режим>:= “BROWSE” – логический параметр, его присутствие указывает, что модуль работает с формами редактирования в виде таблицы, иначе модуль работает с отдельными записями.

<Обновить>:= “REFRESH” – логический параметр, указывает, что после успешного вызова модуля необходимо обновить редактируемый набор данных.

<Описание кнопок вызова надстроек>:= “BTN” “(” {<Свойства кнопки> “,”}* “)”

<Свойства кнопки>:= {“TBL” “=” <Имя таблицы> | <Имя представления>} {“CAPTION” “=”<Текст>} {“HINT” “=” <Текст>} {“INFO” “=” <Текст>} {“GLYPH” “=” <Текст>}

<Текст>:= {<Буква>*|<Цифра>*| “.” | “,”}

Значение параметров:

“TBL” – наименование таблицы или представления, для которой может применяться данный модуль

“CAPTION” – надпись на кнопке вызова модуля;

“HINT” – всплывающая подсказка для кнопки вызова модуля;

“INFO” – дополнительная информация, которая передаётся модулю при данном вызове. Например, для модуля печати отчётов в этой строке может указываться имя файла шаблона.

“GLYPH” – изображение для кнопки вызова модуля.

Включение информации из других спецификаций.

Для использования информации об определения таблиц и представлений из других спецификаций используется следующие предложение:

<Использование спецификаций> := “USES” {<Псевдоним> “=”{“F””.””«”<Имя файла>“»” } “,” {“S””.””«”<Схема>“»”} “,” [“R”] “;”}*

<Псевдоним>:={<Буква>*|<Цифра>*} – псевдоним спецификации. Используется в качестве префикса при ссылках на структуры из данной спецификации.

<Имя файла>:={<Буква>*|<Цифра>* “.”} – имя файла спецификации.

<Схема>:={<Буква>*|<Цифра>* “.”} – переопределение схемы.

Пример:

```
USES ADDR=F."addrs.ini",S."SQLSRV.ADDR2.dbo.";
     UMD=F."UMD2013.ini",S." SQLSRV.UMD_NEW.dbo."
```

Приложение Б. Спецификация АИС «Единый общегородской регистр адресов недвижимости»

АИС «Единый общегородской регистр адресов недвижимости» (АИС ЕОРАН) является базовой для всех систем градостроительного хозяйства, упомянутых в данной работе. Для лучшего понимания разработанной спецификации АИС ЕОРАН приведём диаграмму БД ЕОРАН (рисунок 39).

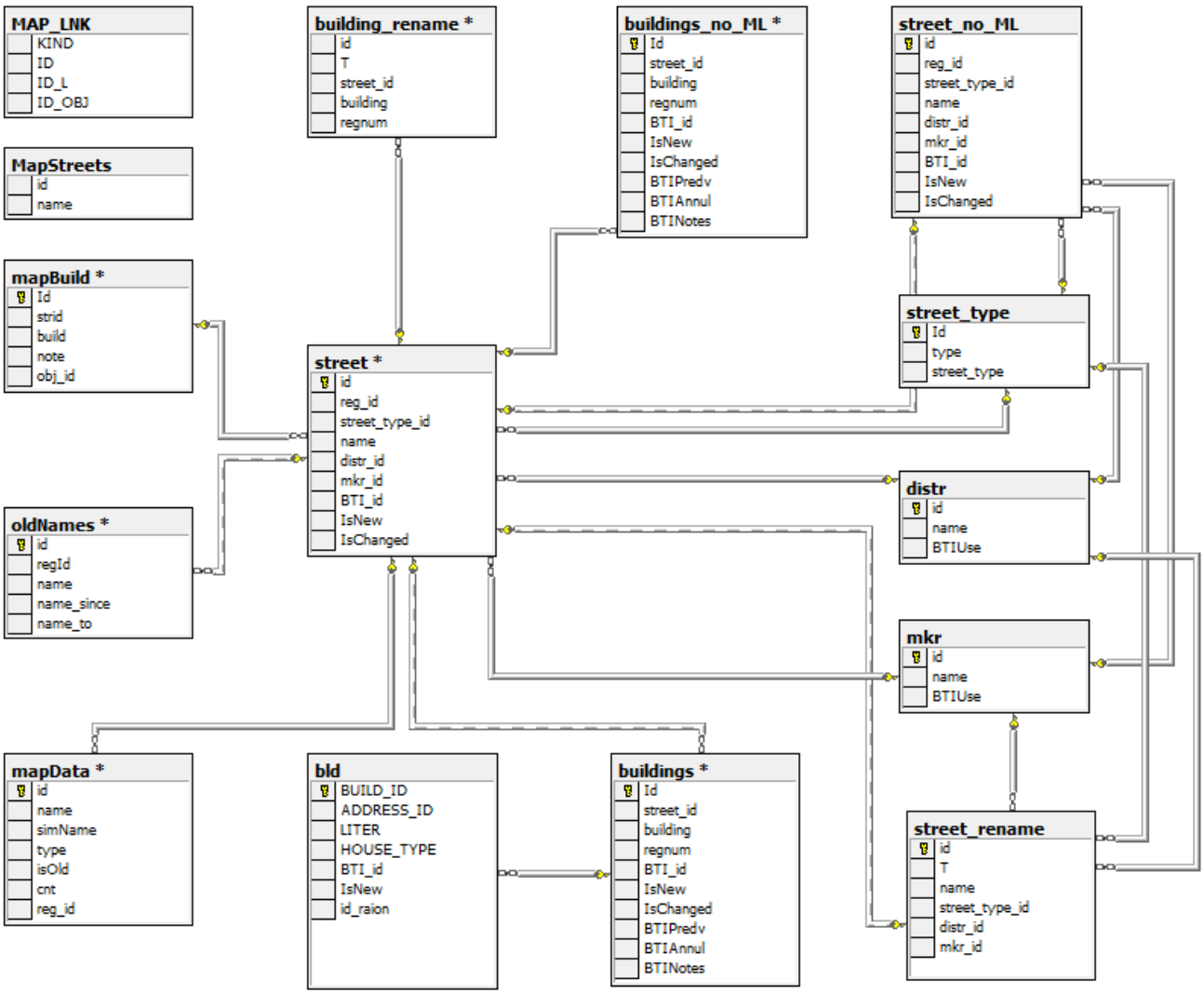

Рисунок 39 Диаграмма БД ЕОРАН

При помощи инструментального средства ГеоАРМ на основе схемы БД ЕОРАН была разработана следующая спецификация ПБД:

```
ADO CONNECTIONSTRING='Provider=SQLOLEDB.1;Persist Security Info=False;User ID=Username;Initial Catalog=addr;Data Source=SRV;Use Procedure for Prepare=1;Auto Translate=True;Packet Size=4096;Use
```

```
Encryption for Data=False;Tag with column collation when
possible=False;'

 ConnectionTimeout=15
 CommandTimeout=30;

CFG DATEFMT=yyyymmdd QuoteTNFmt=.%s UpdateMode=2 GUIStyle=1
TopRefsOnly AppName='АИС ЕОРАН' AppTitle=ЕОРАН GUISTYLE FIXED
UpdateMode=2 TOPREFS;

PLUGIN MAP PATH=\\SRV\prg\GEOARM_PRG\Pass\DBEdit\Rept.dll
INFO=Распоряжение_Шаблон.doc PROC=Init
PLUGIN MAP1 PATH=\\SRV\prg\GEOARM_PRG\Pass\DBEdit\Rept.dll
INFO1=Распоряжение_присоение_адреса.doc PROC=Init;

TABLE buildings FOR dbo.buildings AS Адреса
FIELDS( Id=P,
   street_id=^vstreet(id),
   building=S,
   regnum=S,
   BTI_id=I,
   IsNew=B,
   IsChanged=B)
 REFS('Здания по адресу'=<bld.ADDRESS_ID,
     'Изменения адреса'=<vBuilding_rename.id
     );

TABLE distr FOR dbo.distr AS Районы NAMES="name"
FIELDS( id=P,   name=N);

TABLE MAP_LNK FOR dbo.MAP_LNK AS 'Связь с картой'
FIELDS( KIND=P,
   ID=I,
   ID_L=I,
   ID_OBJ=I);
```

```
TABLE mapBuild FOR dbo.mapBuild AS 'Строения АП' NAMES="build"
FIELDS( Id=P,
   strid=I,
   build=N WIDTH=7,
   note=S WIDTH=30,
   obj_id=I)
 REFS( R=strid^vstreet(reg_id)
);

TABLE mapData FOR dbo.mapData NAMES="name"
FIELDS( id=P,
   name=N WIDTH=50,
   simName=S WIDTH=50,
   type=^vstreet(id),
   isOld=B,
   cnt=I,
   reg_id=I);

TABLE MapStreets FOR dbo.MapStreets AS 'Улицы АП' NAMES="name"
FIELDS( id=P,
   name=N);

TABLE mkr FOR dbo.mkr AS Микрорайоны NAMES="name"
FIELDS( id=P,
   name=N WIDTH=16);

TABLE oldNames FOR dbo.oldNames AS 'Старые названия' NAMES="name"
FIELDS( id=P,
   regId=^vstreet(id),
   name=N WIDTH=50,
   name_since=D,
   name_to=D);

TABLE street FOR dbo.street AS Улицы NAMES="name"
```

```
FIELDS( id=P,
   reg_id=I,
   street_type_id=^street_type(Id),
   name=N WIDTH=50,
   distr_id=^distr(id),
   mkr_id=^mkr(id),
   BTI_id=I,
   IsNew=B,
   IsChanged=B)
 REFS(Адреса=<vbuildings.street_id,Изменения=<vStreet_rename.id
);

TABLE street_type FOR dbo.street_type AS 'Типы улиц' NAMES="type"
FIELDS( Id=P,
   type=N,
   street_type=S);

TABLE HISTORY_OBJ FOR dbo.AP_OBJECTS AS 'Снесенные строения'
FIELDS( ID=P AS №,
   KOD_OBJ=I AS -,
   NAME=S AS Наименование,
   OBJ= AS -,
   KOD_ADR=I AS 'Код Адреса',
   UL=S AS Улица,
   DOM=S AS Дом,
   DATE_UPDATE=D AS 'Дата удаления');

TABLE bld FOR dbo.bld AS Здания
FIELDS( ADDRESS_ID=P^buildings(Id) AS -,
   BUILD_ID=I AS №,
   LITER=S AS Литера,
   HOUSE_TYPE=S AS 'Тип здания',
   id_raion=I);

VIEW vbld FOR bld AS Здания ADDRS="name;building"
```

```
FIELDS(ADDRESS_ID=  AS #,
  BUILD_ID=  AS №,
   ST=ADDRESS_ID.street_id.street_type_id.type  AS 'Тип ул.',
   name=ADDRESS_ID.street_id.name  AS Улица  WIDTH=50,
   Distr=ADDRESS_ID.street_id.distr_id.name  AS Район,
   mk=ADDRESS_ID.street_id.mkr_id.name  AS Микрорайон  WIDTH=16,
   building=ADDRESS_ID.building  AS Дом,
   regnum=ADDRESS_ID.regnum  AS 'Рег. номер дома',
  LITER=  AS Литера,
  HOUSE_TYPE=  AS 'Тип здания')
);

VIEW vbuildings FOR buildings AS Адреса
NAMES="strn,strt,mk,building" ADDRS="strn,building"
FIELDS(Id=  AS -,
   strt=street_id.street_type_id.type  AS Тип,
   strn=street_id.name  AS Улица  WIDTH=50,
   mk=street_id.mkr_id.name  AS Микрорайон  WIDTH=16,
  building=  AS 'Номер дома',
  regnum=  AS 'Номер в реестре',
  BTI_id=  AS 'Код в БТИ',
  IsNew=  AS 'Новый (из БТИ)',
  IsChanged=  AS Изменено)
);

VIEW vstreet FOR street AS Улицы NAMES="name,ST,mk"
FIELDS( Id=id  AS -,
  reg_id=  AS 'Номер в реестре',
   ST=street_type_id.type  AS Тип,
  name=  AS Улица  WIDTH=50,
   Distr=distr_id.name  AS 'Район города',
   mk=mkr_id.name  AS Микрорайон  WIDTH=16,
  BTI_id=  AS 'Код в БТИ',
  IsNew=  AS 'Новая (из БТИ)',
  IsChanged=  AS Изменено)
```

```
);

VIEW voldNames FOR oldNames AS 'Старые названия' NAMES="name"
FIELDS(id=  AS -,
   ctype=regId.street_type_id.type  AS Тип,
   cname=regId.name  AS 'Название в настоящее время'  WIDTH=50,
  name=  AS 'Старое название'  WIDTH=50,
  name_since=  AS 'Название с',
  name_to=  AS 'Название по')
);

VIEW vmapBuild FOR mapBuild AS 'Строения АП' NAMES="build"
FIELDS(Id=  AS -,
   U=R.name  AS Улица  WIDTH=50,
  build=  AS 'Номер строения'  WIDTH=7,
  note=  AS Примечания  WIDTH=30,
  obj_id=  AS 'Номер объекта на карте')
);

VIEW vmapData FOR mapData NAMES="name"
FIELDS(id=  AS -,
   t=type.street_type_id.type  AS Тип,
  name=  AS Улица  WIDTH=50,
  simName=  AS 'Улица из АР если нет совпадения'  WIDTH=50,
  cnt=  AS -)
);

TABLE buildings_no_ML FOR dbo.buildings_no_ML AS 'Адреса не
привязанные к карте' READONLY
FIELDS( Id=P,
   street_id=^vstreet(id),
   building=S,
   regnum=S);
```

```
TABLE street_no_ML FOR dbo.street_no_ML AS 'Улицы не привязанные к
карте' NAMES="name" READONLY
FIELDS( id=P,
   reg_id=I,
   street_type_id=^street_type(Id),
   name=N WIDTH=50,
   distr_id=^distr(id),
   mkr_id=^mkr(id));

VIEW vbuildings_no_ML FOR buildings_no_ML AS 'Адреса не привязанные
к карте' READONLY
FIELDS(Id=  AS -,
   strt=street_id.street_type_id.type  AS Тип,
   strn=street_id.name  AS Улица  WIDTH=50,
   mk=street_id.mkr_id.name  AS Микрорайон  WIDTH=16,
  building=  AS 'Номер дома',
  regnum=  AS 'Номер в реестре')
);

VIEW vstreet_no_ML FOR street_no_ML AS 'Улицы не привязанные к
карте' NAMES="name" READONLY
FIELDS( Id=id  AS -,
  reg_id=  AS 'Номер в реестре',
   ST=street_type_id.type  AS Тип,
  name=  AS Улица  WIDTH=50,
   Distr=distr_id.name  AS 'Район города',
   mk=mkr_id.name  AS Микрорайон  WIDTH=16)
);

TABLE street_rename FOR dbo.street_rename AS 'Переименование улиц'
NAMES="name" READONLY
FIELDS( id=P^street(id),
   T=P,
   name=N,
   street_type_id=^street_type(Id),
```

```
   distr_id=^distr(id),
   mkr_id=^mkr(id));

TABLE building_rename FOR dbo.building_rename AS 'Переименование
адресов' READONLY
FIELDS( id=P^buildings(Id),
   T=P,
   street_id=^street(id),
   building=S,
   regnum=S);

VIEW vStreet_rename FOR street_rename AS 'Переименование улиц'
NAMES="name" READONLY
FIELDS(id=  AS -,
  T=  AS 'Дата изм.',
  name=  AS Улица,
   st=street_type_id.type  AS 'Тип ул.',
   distr=distr_id.name  AS Район,
   mkr=mkr_id.name  AS Микрорайон  WIDTH=16)
);

VIEW vBuilding_rename FOR building_rename AS 'Переименование
адресов' READONLY
FIELDS(id=  AS -,
  T=  AS 'Дата изм.',
   street=street_id.name  AS Улица  WIDTH=50,
   type=street_id.street_type_id.type  AS 'Тип ул.',
   distr=street_id.distr_id.name  AS Район,
   mkr=street_id.mkr_id.name  AS Микрорайон  WIDTH=16,
  building=  AS Дом,
  regnum=  AS 'Рег. номер')
);

TABLE APDat_Addrs FOR APDat.dbo.AddrsEx READONLY
FIELDS( id=P,
```

```
   id_s=^APDat_streets(id),
   Num=S,
   Miss=B)
 REFS(Объекты=<APDat_PanAddrs.N
);

TABLE APDat_PanAddrs FOR APDat.dbo.PanAddrs READONLY
FIELDS( N=P^APDat_Addrs(id),
   Pan=I);

TABLE APDat_streets FOR APDat.dbo.streetsEx NAMES="Name" READONLY
FIELDS( id=P AS №,
   Name=N AS Наименование,
   Miss=B AS 'Нет в БД')
 REFS(Адреса=<vAPDat_Addrs.id_s
);

VIEW vAPDat_Addrs FOR APDat_Addrs NAMES="Street,Num" READONLY
FIELDS(id=  AS №,
  id_s=  AS -,
   Street=id_s.Name  AS Улица,
  Num=  AS Дом,
  Miss=  AS 'Нет в БД')
);

TBL_MENU (
P1:Адресный реестр
.vstreet=Улицы
.voldNames='Старые названия'
.vbuildings=Адреса
.vbld=Здания
P2:Адресный план
.vmapData=Улицы
.vmapBuild=Строения
P3:История
```

```
.HISTORY_OBJ='Снесенные здания'
P4:Статистика
.vstreet_no_ML='Улицы непривязанные к карте'
.vbuildings_no_ML='Здания непривязанные к карте'
P5:Карта
.APDat_streets=Улицы
.vAPDat_Addrs=Адреса
);
```

В результате интерпретации приведённой спецификации в ГеоАРМ автоматически создаются следующие экранные формы:

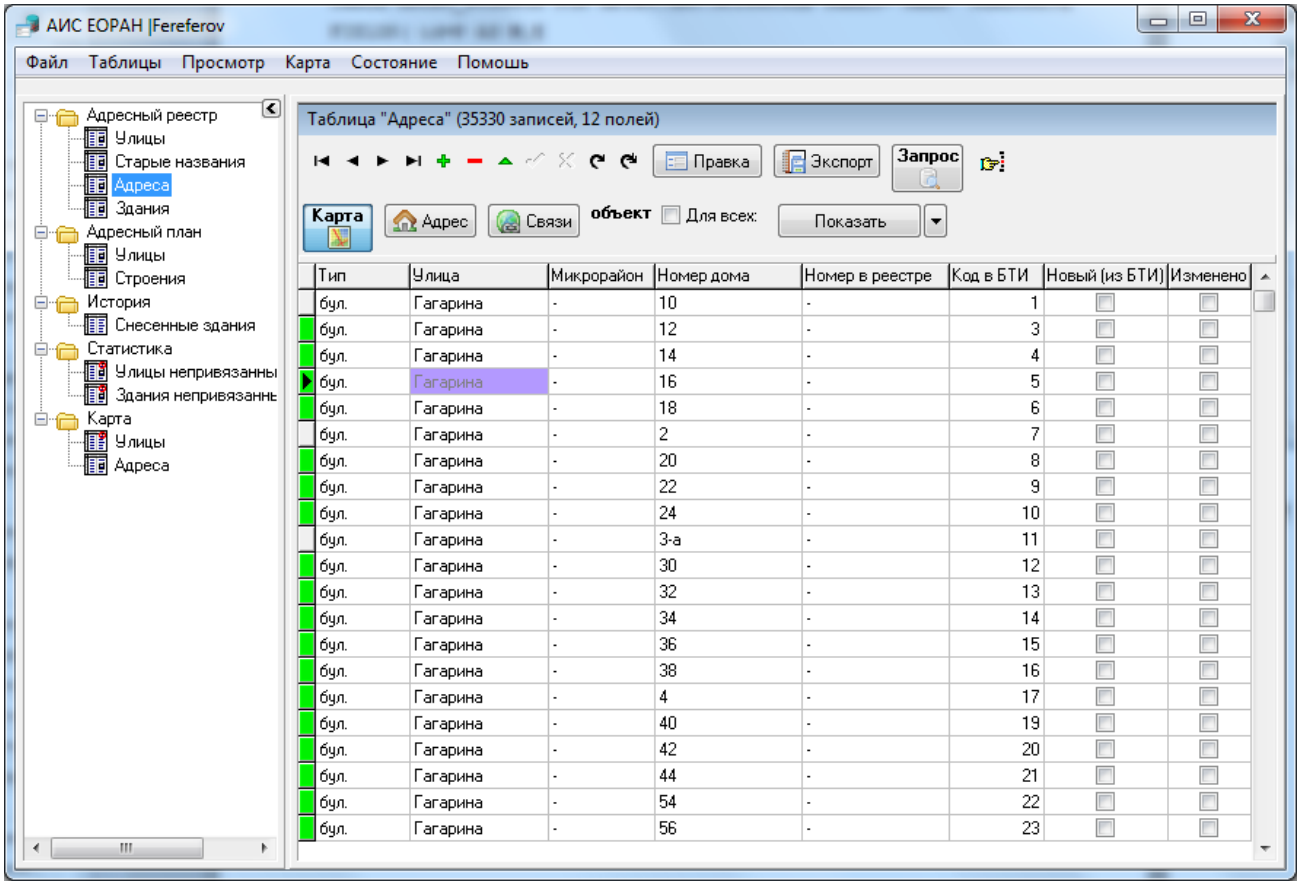

Рисунок 40 Представление "Адреса" в табличном режиме

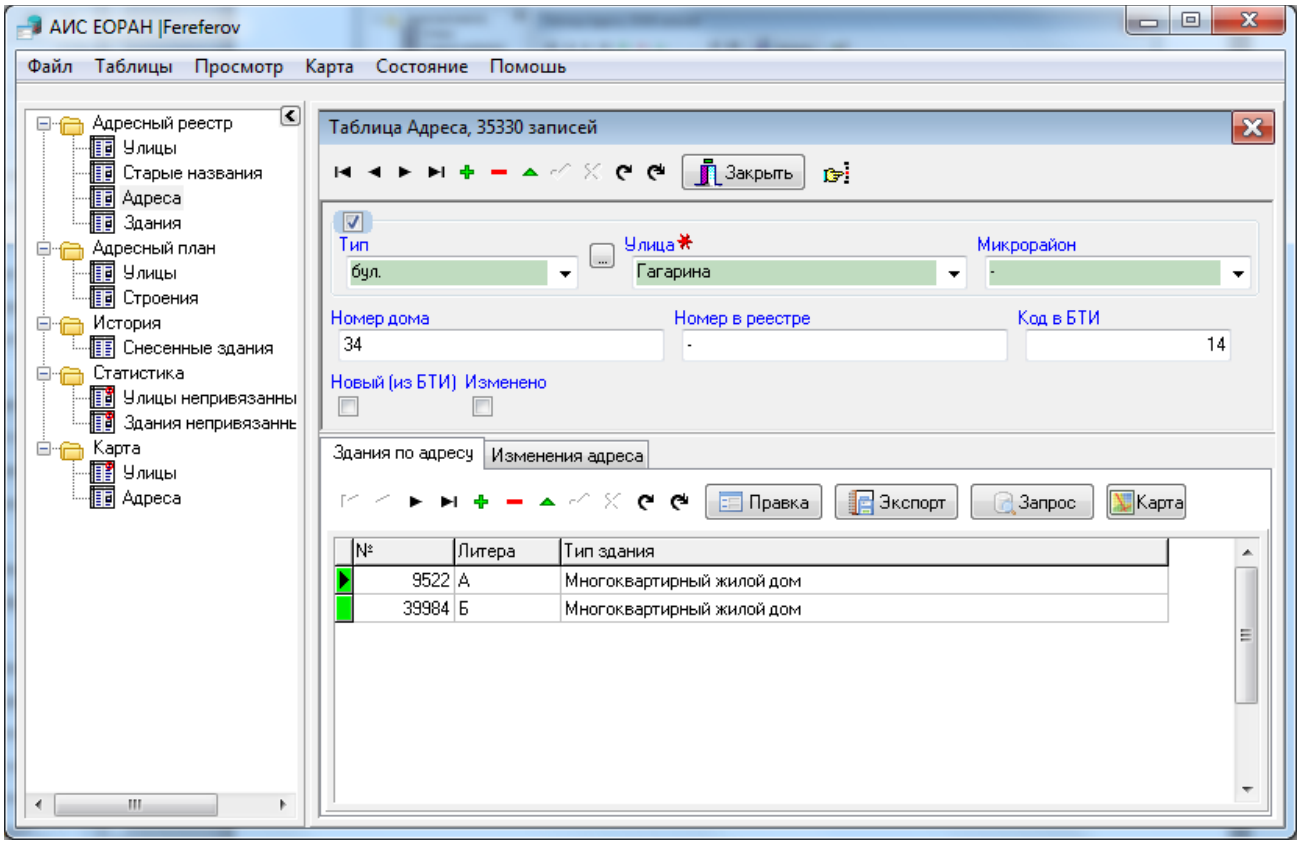

Рисунок 41 Представление "Адреса" в режиме формы

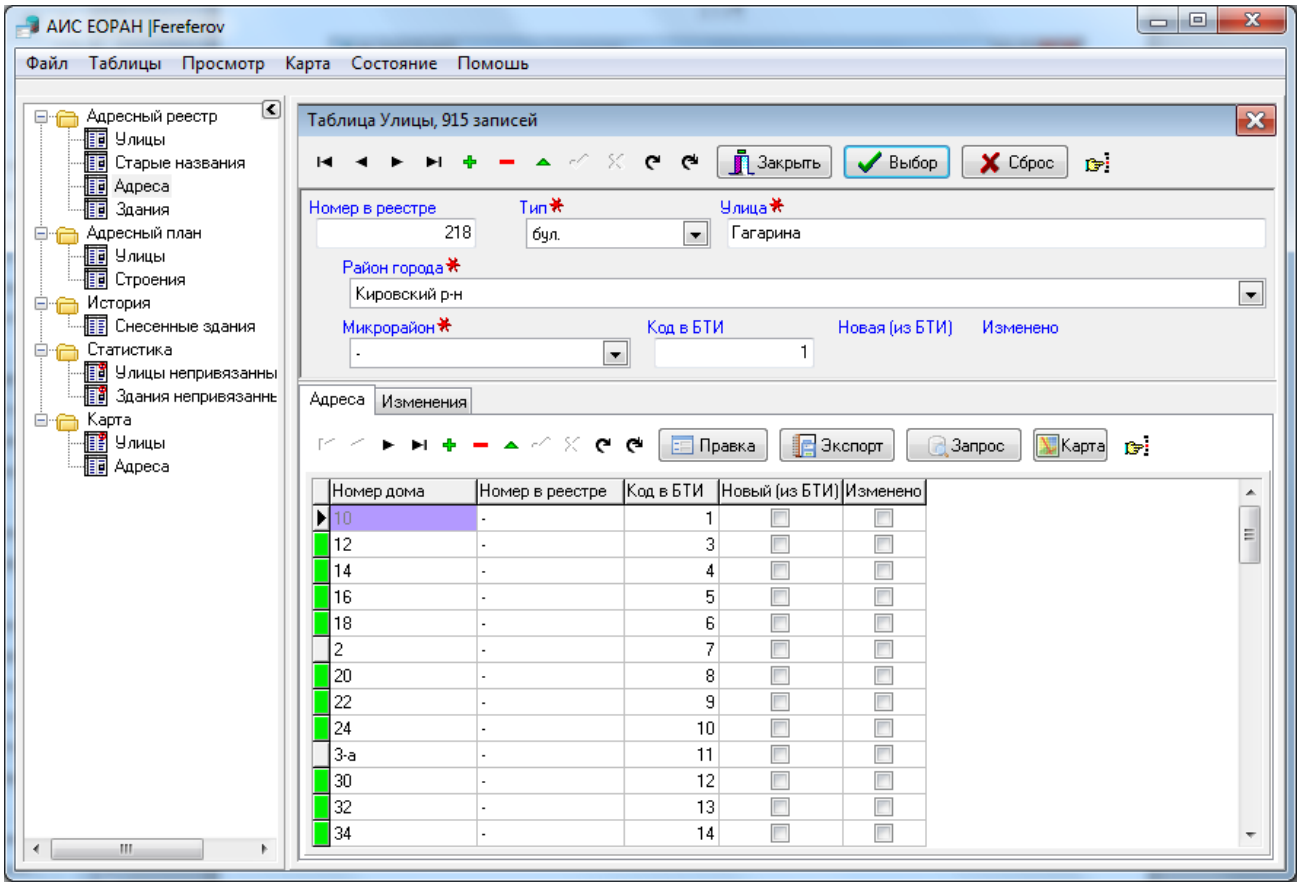

Рисунок 42 Представление "Улицы" в режиме формы

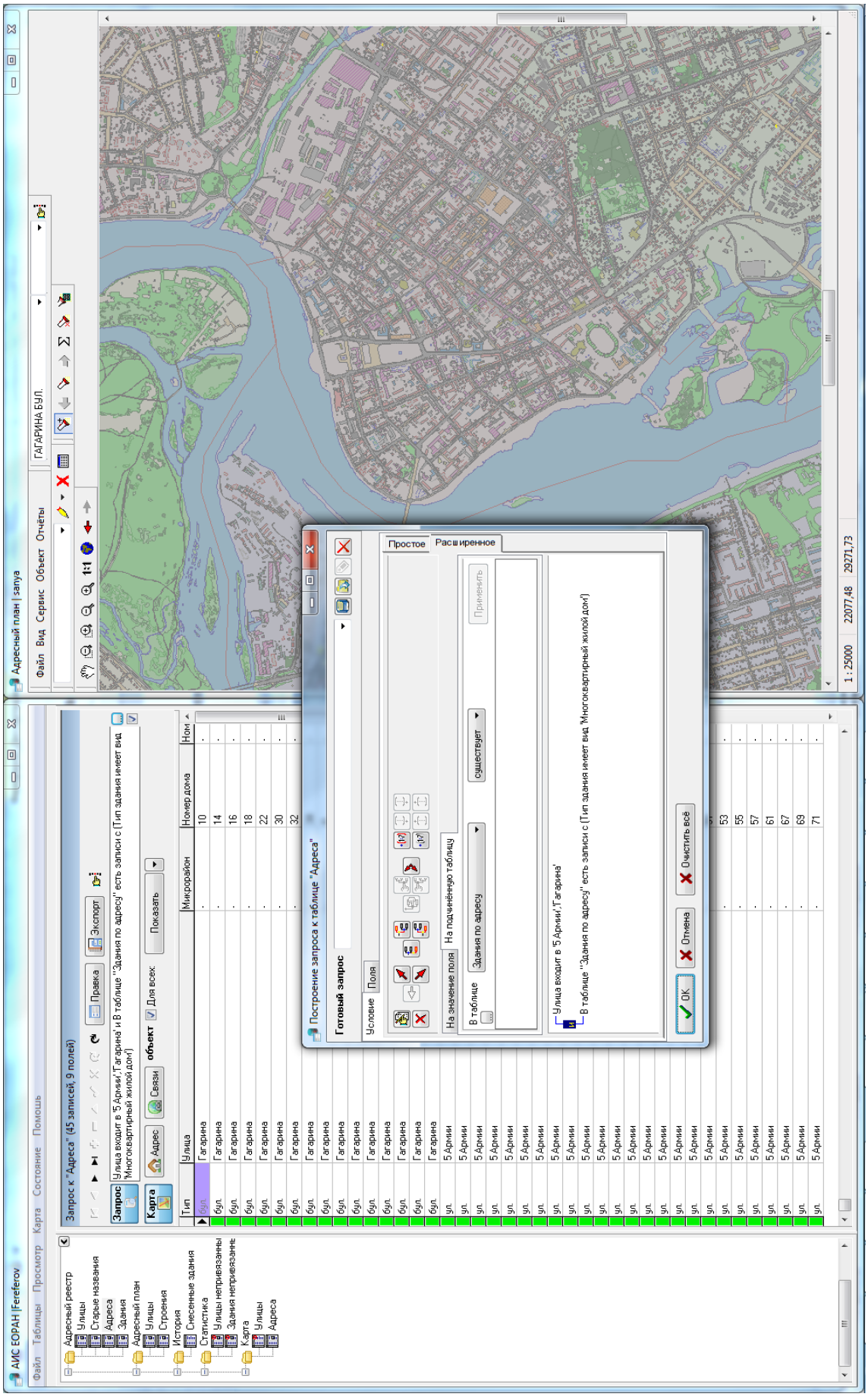

Рисунок 43 АИС ЕОРАН