МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение высшего
профессионального образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

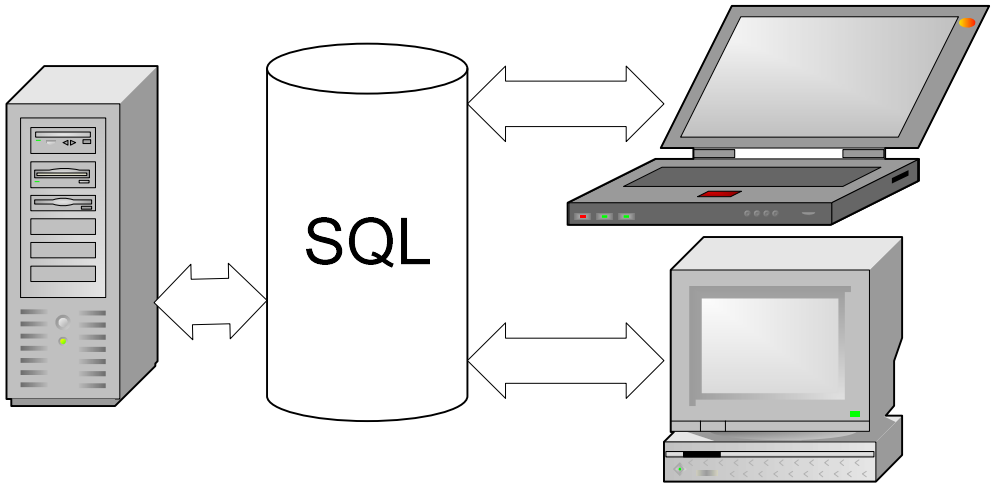
Физический факультет

Кафедра информационных технологий в физических исследованиях

С.А. Минеев
Ю.Е. Чуманкин

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММ, ВЗАИМОДЕЙСТВУЮЩИХ С БАЗАМИ ДАННЫХ

Учебно-методическое пособие



Нижний Новгород
2018

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского

Физический факультет

Кафедра информационных технологий в физических исследованиях

С.А. Минеев
Ю.Е. Чуманкин

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММ, ВЗАИМОДЕЙСТВУЮЩИХ С БАЗАМИ ДАННЫХ

Учебно-методическое пособие

Рекомендовано методической комиссией физического факультета
для студентов ННГУ, обучающихся по направлению подготовки
09.03.02 «Информационные системы и технологии»

Нижний Новгород
2018

УДК 004.67(072)
ББК Ч 426.29(07)
М-57

М-57. Минеев С. А., Чуманкин Ю.Е. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММ, ВЗАИМОДЕЙСТВУЮЩИХ С БАЗАМИ ДАННЫХ. Учебно-методическое пособие в электронной форме. – Нижний Новгород. Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского, 2018. – 66 с.

Рецензент: к.ф.-м.н., старший научный сотрудник
Будников Н.С.

В пособии изложены основные архитектурные принципы создания приложений, взаимодействующих с реляционными базами данных, приведены практические рекомендации по разработке и отладке таких приложений. В пособие включен комплект учебных задач для организации практических занятий по курсу “Управление данными”.

УДК 004.67(072)
ББК Ч 426.29(07)

Оглавление

1 Введение

Современные информационные системы немыслимы без применения реляционных баз данных (далее БД) и соответствующих систем управления (далее СУБД). Хотя нереляционные СУБД (иерархические, объектно-ориентированные, гибридные и др.) тоже находят свое применение при построении информационных систем, но первенство по производительности и объемам хранимой/обрабатываемой информации прочно удерживается реляционными СУБД уже более 20 лет. Основу долголетия реляционной технологии хранения/обработки данных составляют: развитая теория отношений (реляционная алгебра), базирующийся на ней уровень абстракции данных и международная стандартизация.

За разнообразием программного инструментария для создания и сопровождения программных комплексов, взаимодействующих с реляционными БД, скрывается тот факт, что базовые технологии даже у продуктов различных производителей очень близки, если не тождественны. Следствия из данного факта следующие: знание теории отношений и уровня абстракции данных – значительное подспорье при освоении различных СУБД и переходе с одной СУБД на другую, архитектура большинства приложений, взаимодействующих с реляционными БД, очень схожа, поддается систематизации и планированию.

Целью данного пособия является помощь студентам, изучающим курс “Управление данными”, при выполнении практических работ по курсу. В пособии изложены основные архитектурные принципы создания приложений, взаимодействующих с реляционными базами данных, приведены практические рекомендации по разработке и отладке таких приложений. В пособие включен комплект учебных задач для организации практических занятий по курсу “Управление данными”.

2 Технологии доступа к БД из программного кода

Современные СУБД скрывают от разработчика программ реальный формат хранящихся в БД данных за “фасадом” абстракции уровня данных. Разработчик программы, взаимодействующей с реляционной БД, оперирует не низкоуровневыми абстракциями “файл”, “поток ввода/вывода”, “признак конца файла”, “буфер чтения” и т. п., а абстракциями, пришедшими из теории отношений - “таблица”, “запись”, “ограничение”, “объединение”. Работу на данном уровне абстракции обеспечивает специальная программная прослойка между программой и собственно БД. Оформляется такая прослойка обычно в форме динамической библиотеки (см. рис. 1). Интерфейс такой библиотеки полностью определяется поставщиком СУБД, но существует целый ряд стандартов, например, ODBC (Open Data Base Connectivity), которые накладывают жесткие ограничения на программный интерфейс к БД. Поставщики СУБД нередко поставляют несколько альтернативных программных интерфейсов, среди которых обязательно есть стандартизированные.

Далее в данном разделе будут рассмотрены несколько промышленных стандартов на программные интерфейсы к БД. Библиотеки, реализующие такие интерфейсы, далее именуются как “драйверы БД”.

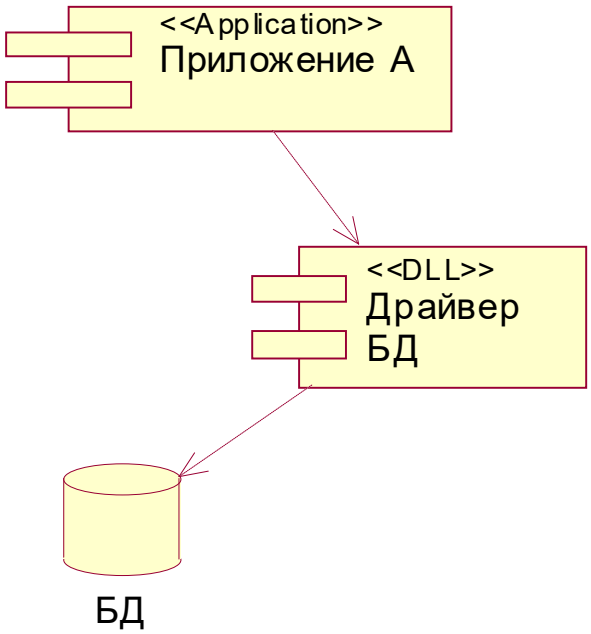


Рисунок 1. Взаимодействие приложения с БД

2.1 Драйверы ODBC

Open Database Connectivity (ODBC) представляет собой спецификацию программного интерфейса для доступа к базам данных. Этот интерфейс основан, в свою очередь, на спецификациях Call-Level Interface (CLI) от X/Open и ISO/IEC. В качестве языка доступа к базам данных применяется Structured Query Language (SQL).

ODBC разработан для максимальной унификации способа взаимодействия приложений с БД различных производителей, то есть одна прикладная программа может без изменения своего исходного текста работать через ODBC-интерфейс с любой БД.

Архитектура ODBC включает следующие компоненты (см. рис.2):

1 **Приложение**. Выполняет прикладные задачи, вызывает функции драйвера ODBC для передачи SQL- выражений и получения результатов.

2. **Менеджер драйверов**. Загружает драйвера по требованию приложения.

3. **Драйвер**. Обрабатывает вызовы функций ODBC специфичным для конкретной СУБД образом и возвращает результат приложению. Если необходимо, драйвер модифицирует запросы в соответствии с особенностями СУБД.

4. **Источник данных**. Хранилище данных. Может размещаться как локально, так и удаленно (в последнем случае на вычислительном узле, где выполняется приложения пользователя, должна быть размещена информация о местоположении БД).

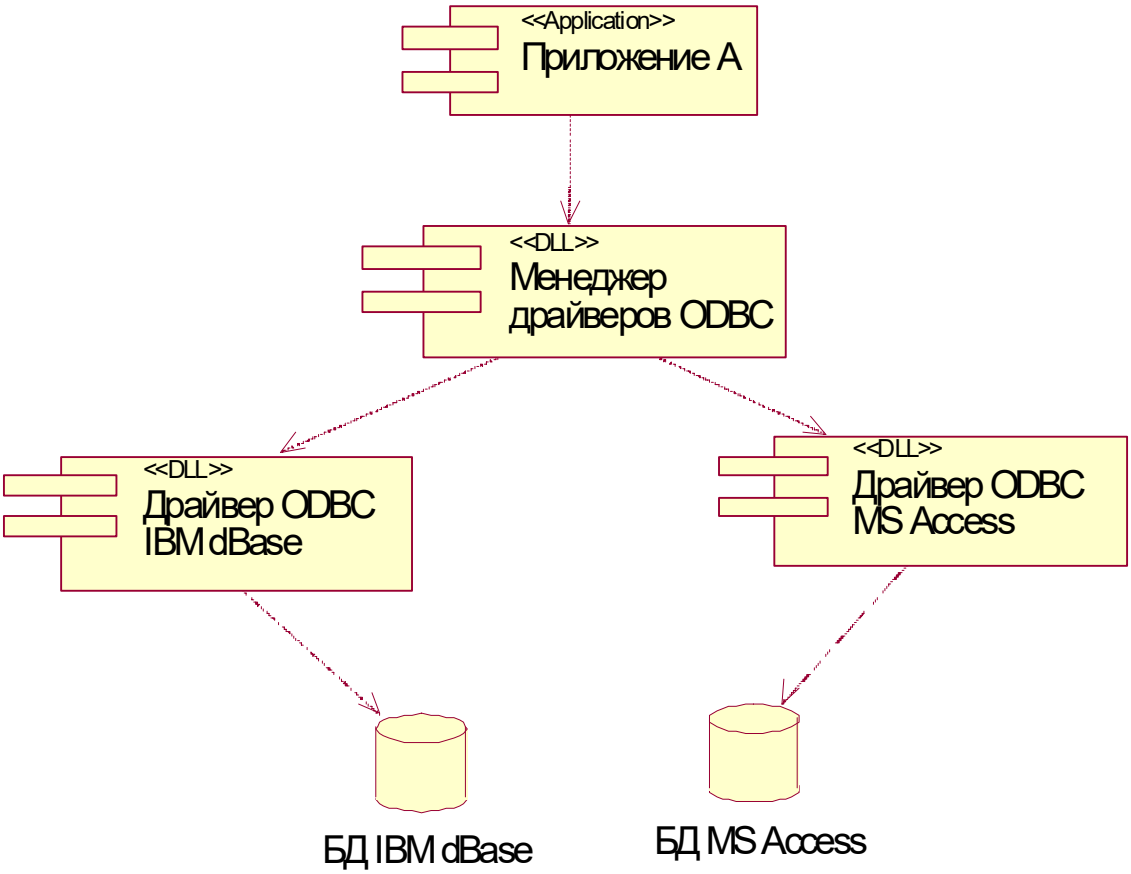


Рисунок 2. Взаимодействие приложения с БД
через ODBC драйверы

Приложение, использующее интерфейс ODBC, выполняет следующие задачи:

- запрашивает соединение с источником данных;
- посылает SQL- запросы к источнику данных;
- описывает область хранения и формат для результатов SQL- запросов;
- запрашивает данные;
- обрабатывает ошибки;
- если необходимо, оповещает пользователя об ошибках;
- осуществляет фиксацию или откат действий в режиме транзакций;
- закрывает соединение с источником данных.

Диспетчер драйверов, поставляемый фирмой Microsoft®, является динамически подключаемой библиотекой (DLL). Основной задачей диспетчера является загрузка драйверов. Дополнительно он выполняет следующие функции:

- Использует файл odbc.ini или системный реестр для установки соответствия между наименованием источника данных и DLL драйвера.

- Обрабатывает несколько инициализирующих вызовов ODBC.

- Обеспечивает доступ ко всем функциям ODBC в каждом драйвере.

- Проводит контроль параметров и последовательности вызовов функций ODBC.

Диспетчер драйверов загружает драйвер нужной БД, когда приложение вызывает функции `SQLBrowseConnect()`, `SQLConnect()` или `SQLDriverConnect()`.

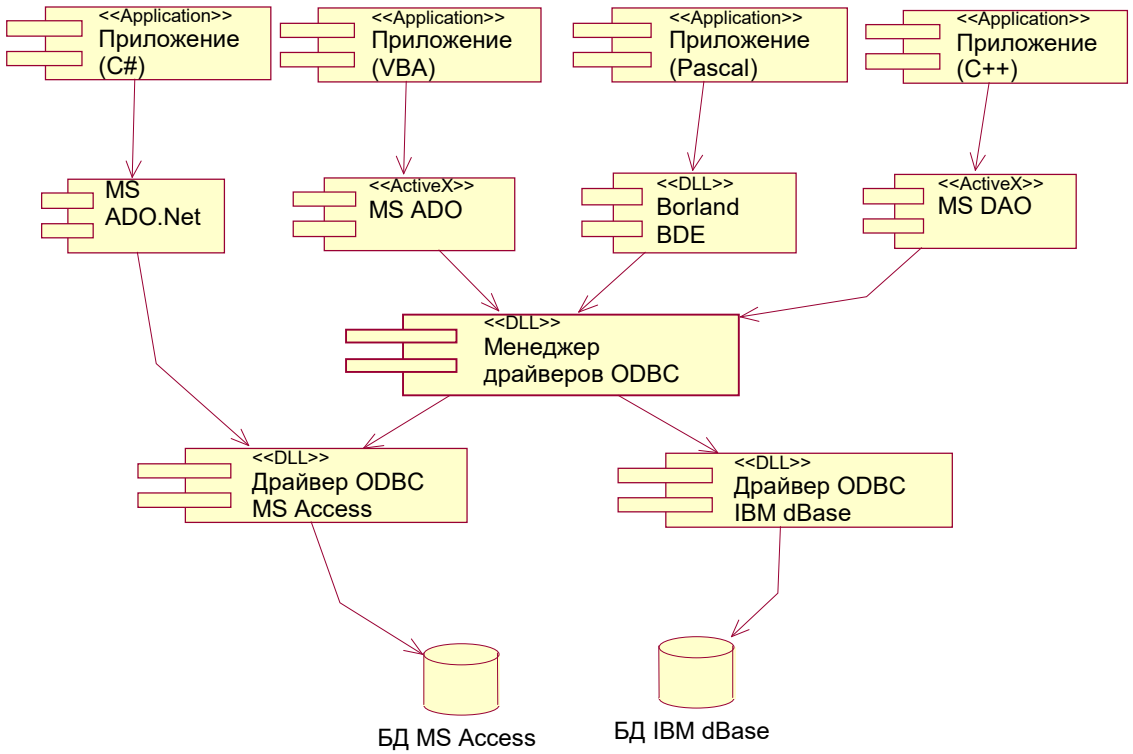


Рисунок 3. Взаимодействие приложения с БД через языковые адаптеры

Прикладные программы редко взаимодействуют с менеджером ODBC напрямую (см. рисунок 2). Исключениями являются инструментальные программы для обслуживания БД, например, Server Explorer для Microsoft Visual Studio и, приложения, к которым предъявляются жесткие требования по размеру и скорости выполнения. Для удобства прикладных программистов постав-

щики сред разработки предлагают дополнительные уровни абстракции, оптимизированные под конкретные языки программирования, например, Microsoft Data Access Objects (Visual Basic, C++), Borland Database Engine (Pascal, C++), Microsoft ActiveX Data Objects (Visual Basic, C++) и Microsoft Access Data Objects .Net (Visual Basic .Net, Managed C++, C#). Далее такие прослойки именуются языковыми адаптерами.

Из приведенной на рисунке 3 схемы следует, что большинство технологий доступа к БД из приложений базируются на драйверах ODBC. Интерфейс ODBC стандартизирован в группе стандартов SQL 2003. Следует отметить, что хотя принципиальных препятствий для применения стандарта ODBC на различных платформах нет, ODBC драйверы распространены только на платформе Microsoft Windows. На платформах Unix-like ODBC-драйверы применяются редко.

2.2 Драйверы JDBC

Назначение и архитектурная организация драйверов JDBC (Java Database Connectivity) аналогичны назначению и архитектурной организации драйверов ODBC. Основное отличие – ориентация на создание приложений на платформе Java. Нередки драйвера JDBC, реализованные как Java-“обертки” для драйверов ODBC. На рисунке 4 приведена диаграмма компонентов для Java-приложений, взаимодействующих с БД.

Неоспоримым преимуществом интерфейса JDBC является его независимость от процессора и операционной системы. Хорошо спроектированный объектно-ориентированный программный интерфейс JDBC (API JDBC) делает излишним использование каких-либо языковых адаптеров.

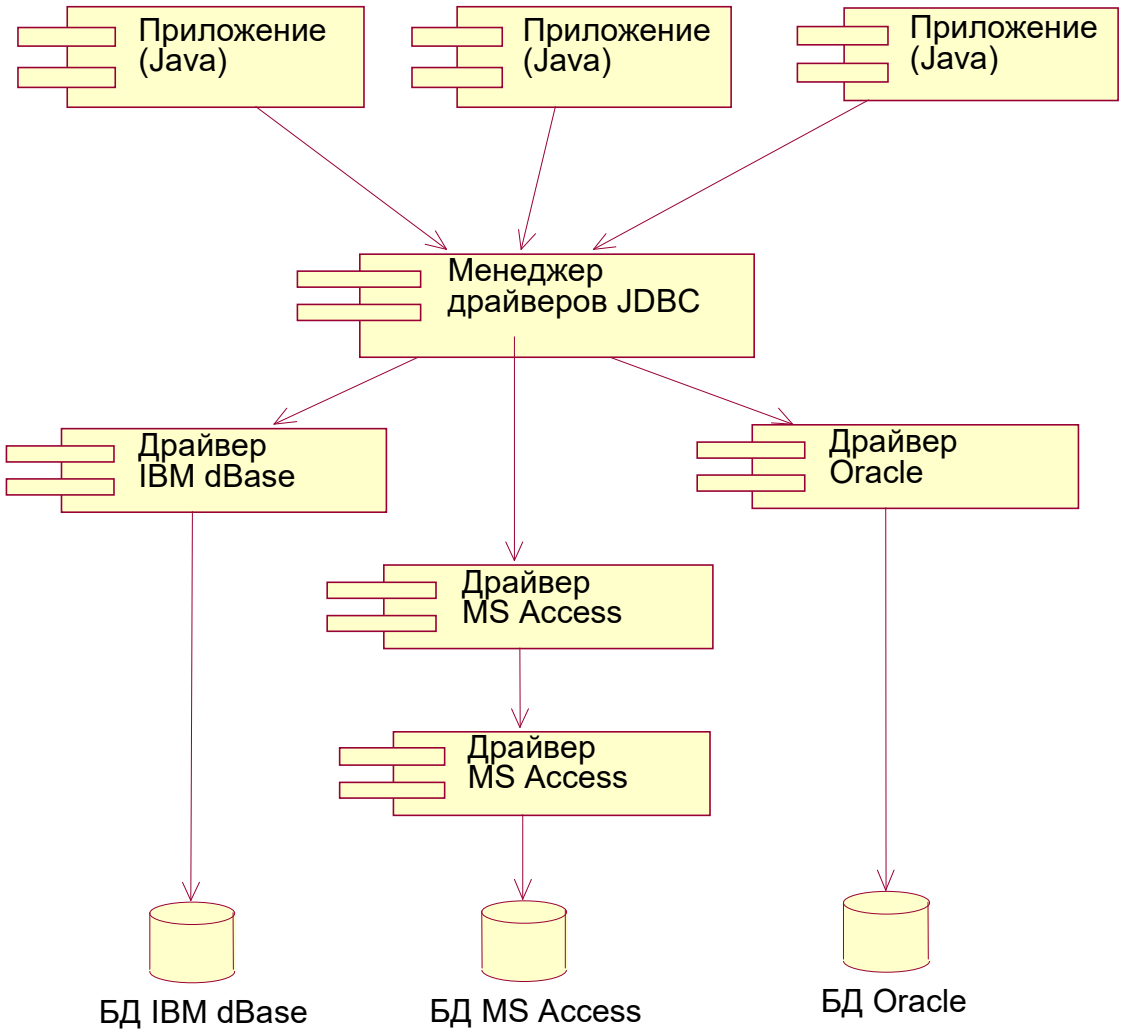


Рисунок 4. Взаимодействие приложений с БД через JDBC драйверы

3 Структура программ, взаимодействующих с БД

3.1 Одноуровневая модель

Если вся логика приложения сосредоточена в одном модуле (интерфейс оператора, вычислительные процедуры и связь с БД), то такое приложение называют “одноуровневым” (single-tier). Благодаря современным средствам обеспечения доступа к БД (см. предыдущий раздел) одноуровневые приложения могут взаимодействовать не только с локальными БД, но и с удаленными, например, серверами SQL. Иногда одноуровневые приложения называют “толстыми” клиентами.

Основные области применения одноуровневых приложений - это простые приложения, использующие локальные БД и приложения, выполняющие интенсивные вычисления, но не часто обращающиеся БД. При создании таких приложений следует помнить, что установление соединения с БД требует времени, а само соединение это ресурс, который необходимо освобождать, если он уже не нужен.

3.2 Двухуровневая модель

В двухуровневых моделях (two-tier) разделяют уровни представления и бизнес-логики. На уровень представления возложена задача организации интерактивного взаимодействия с оператором. За счет минимизации вычислительной нагрузки получаются очень компактные и нетребовательные к ресурсам приложения, которые могут выполняться на мобильных ПЭВМ, смартфонах и др. Часто такие приложения называют “тонкими” клиентами. Основной недостаток тонких клиентов – требование постоянного подключения к каналу передачи данных для выполнения своих функций.

Уровень бизнес-логики нагружен вычислительными задачами на основе хранящихся в БД данных. Запросы на запуск вычислений поступают со стороны тонких клиентов. Строго говоря, приложением, взаимодействующим с БД, в двухуровневой модели выступает только сервер бизнес-логики. Программа-клиент может даже и не “знать” с какой БД идет работа.

Пример связи компонентов в двухуровневой модели приведен на рис. 5.

Сейчас двухуровневые модели получили широкое распространение в Internet. Программу клиента (загружаемый апплет) исполняет Web-браузер, а сервер бизнес-логики содержит локальную БД, что удешевляет разработку и экономит вычислительные ресурсы.

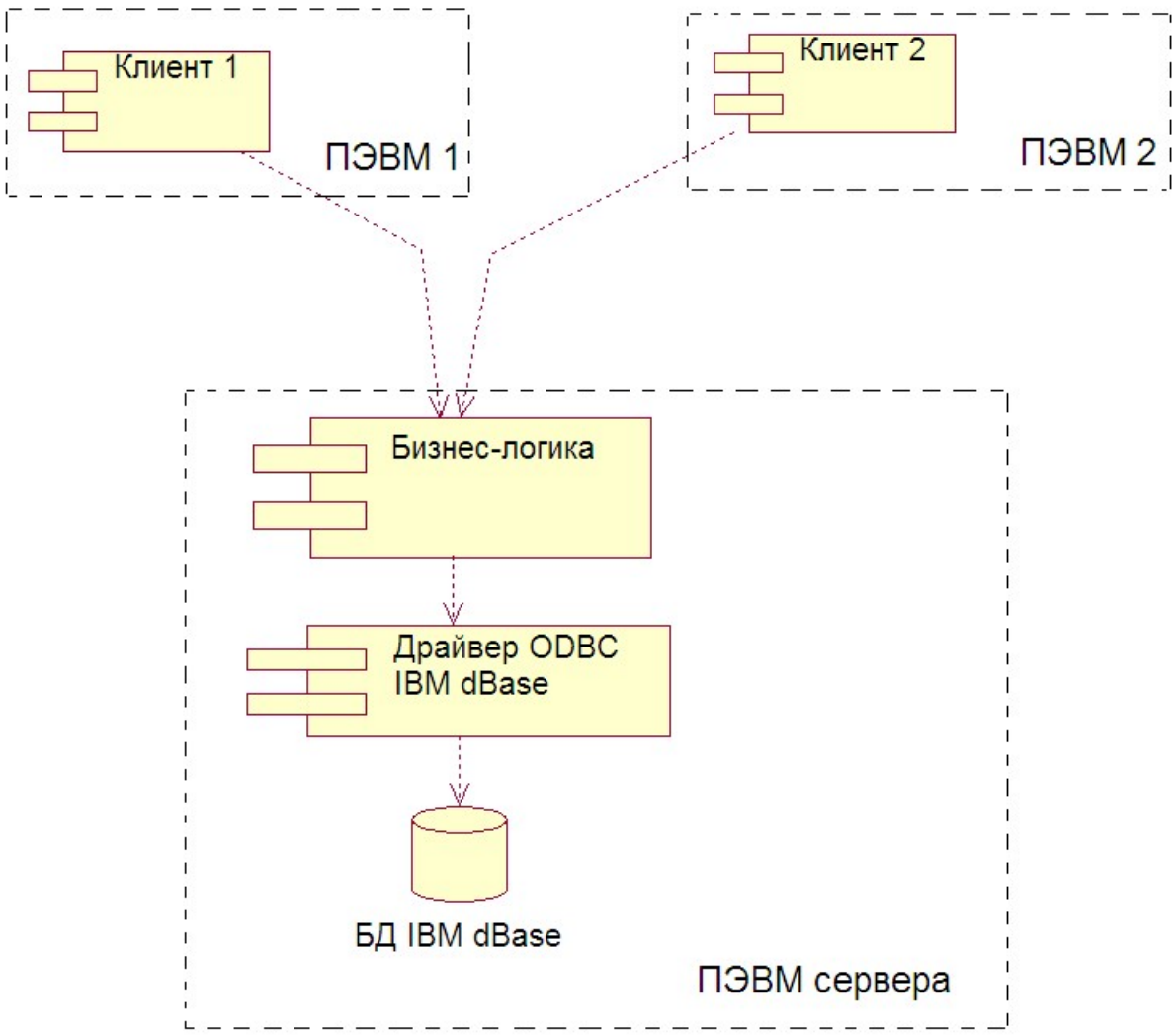


Рисунок 5. Связь компонентов в двухуровневой модели

3.3 Трехуровневая модель

Крупные корпоративные информационные системы часто сталкиваются с ограничениями, связанными с недостаточной производительностью вычислительных узлов и пропускной способностью связных каналов. Для обеспечения масштабируемости таких систем используют трехуровневую и даже n-

уровневые модели приложений. Разработчики информационных систем сознательно идут на усложнение (а значит и на удорожание) как самих приложений, так и вычислительной инфраструктуры.

В трехуровневых и многоуровневых системах выделяют уровни представления, бизнес-логики и хранения данных. Каждый из уровней размещают на различных вычислительных узлах. Уровень представления наименее подвержен изменениям при масштабировании, причинами которого могут быть увеличение числа клиентов (увеличиваются коммуникационные затраты уровня бизнес-логики), увеличение числа задач, решаемых на уровне бизнес-логики (увеличиваются вычислительные затраты серверов БД). N-уровневая архитектура позволяет добавлять ресурсы для устранения узких мест. Достаточно просто решаются задачи резервирования, обслуживания аппаратных и программных средств. Замена программного обеспечения и аппаратных средств на любом из уровней может быть произведена независимо. Пример структуры трехуровневой корпоративной системы приведен на рис. 6.

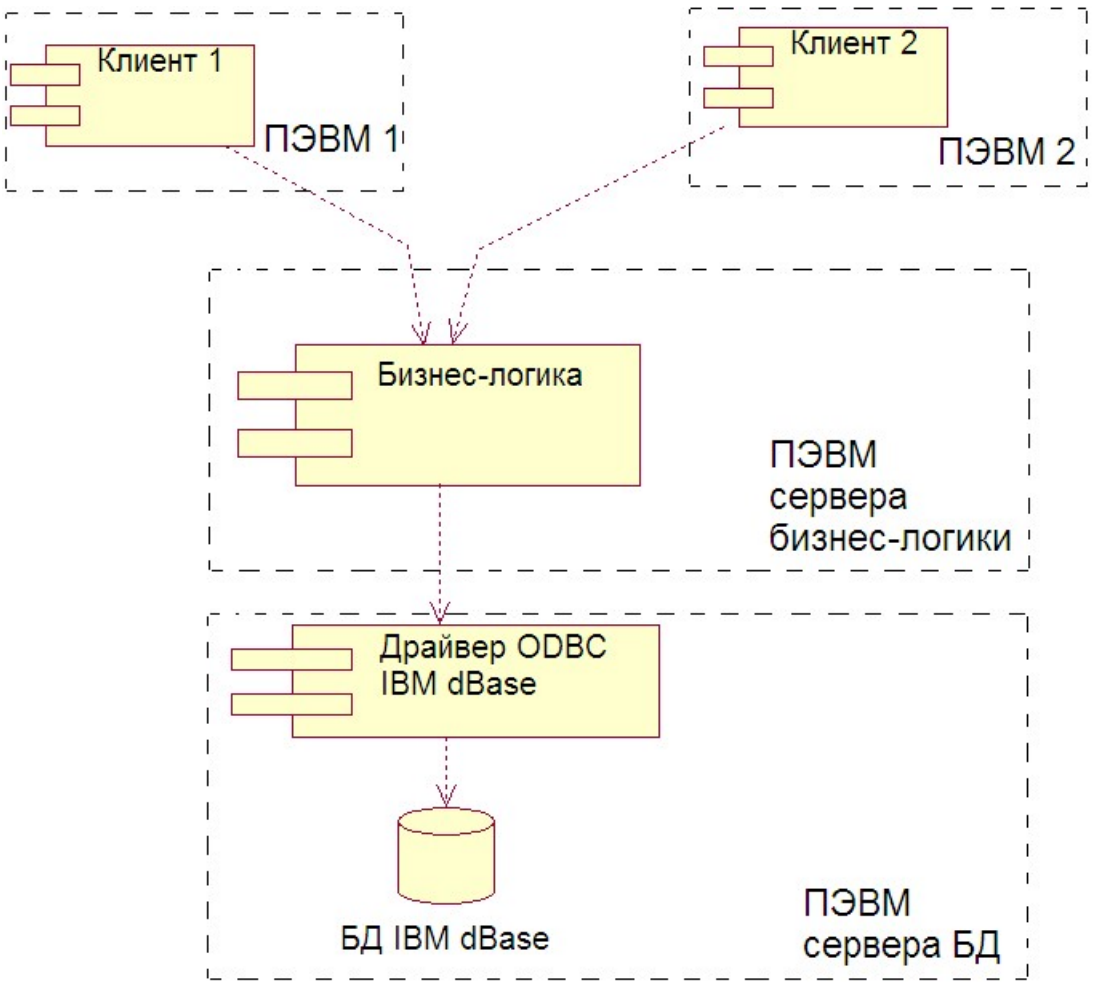


Рисунок 6. Связь компонентов в трехуровневой модели

4 Язык запросов SQL

Язык запросов SQL (Structured Query Language) состоит из двух подмножеств — языка описания данных DDL (Data Definition Language) и языка обработки данных DML (Data Manipulation Language).

Язык описания данных, это язык, используемый для описания атрибутов базы данных, особенно таблиц, полей, индексов и способов сохранения данных. ANSI определяет конструкции DDL как имеющие ключевые слова CREATE, DROP и ALTER. Его рассмотрение выходит за рамки данного пособия.

Язык обработки данных, это язык, используемый для манипуляции данными, добавления, изменения, удаления и выборки данных из таблиц реляционной БД. Основным средством, предоставляемым DML, являются следующие инструкции INSERT, UPDATE, DELETE, SELECT. Перед тем, как подробно рассмотреть эти инструкции введем базу данных из двух таблиц для иллюстрации примеров.

База данных представляет собой хранилище приходно-расходных операций. В ней есть две таблицы, первая — табл. 1 представляет собой список некоторых агентов, которые участвуют в операциях, вторая —

табл. 2, представляет собой список операций, каждая операция случается в определенный момент времени, характеризуемый датой, в операцию вовлечен всегда какой-нибудь агент и операция связана с тем, что на некоторую сумму изменился счет.

Табл. 1. tblAgent

Имя поля	**Тип данных**	**Комментарий**
ID	Счетчик	Ключевое поле, уникальный идентификатор каждой записи,
Name	Текстовый	Имя агента, с которым связана операция
Notes	Текстовый	Характеристика агента

Табл. 2. tblTransaction

Имя поля	Тип данных	Комментарий
ID	Счетчик	Ключевое поле, уникальный идентификатор каждой записи,
Date	Дата	Дата совершения операции
AgentID	Числовой	Идентификатор агента, участвующего в данной операции
Amount	Денежный	Сумма операции

Инструкция INSERT

Добавляет запись или записи в таблицу. Эта инструкция образует запрос на добавление записей.

Синтаксис

Запрос на добавление одной записи:

INSERT INTO назначение [(поле_1[, поле_2[, ...]])]
VALUES (значение_1[, значение_2[, ...])

Параметры этого запроса перечислены ниже:

- назначение — имя таблицы, в которую добавляются данные;
- поле_1, поле_2 — имена полей, в которые добавляются данные;
- значение_1, значение_2 — значения, которые должны быть добавлены.

Следующий пример добавляет в таблицу агентов запись об одном агенте.

Пример 1

```
INSERT INTO  tblAgent(ID, Name, Notes)
             VALUES (1001, 'Агент Б', 'Надежный агент')
```

Следует отметить, что если в таблице есть ключевое поле и оно имеет тип счетчик, то тогда его в инструкции INSERT можно опустить, значение этому полю будет задано автоматически.

Пример 2

```
INSERT INTO  tblAgent (Name, Notes)
             VALUES ('Агент А', 'Отличный агент')
```

Инструкция UPDATE

Создает запрос на обновление, который изменяет значения полей указанной таблицы на основе заданного условия отбора.

Синтаксис

UPDATE таблица

SET новое_значение

WHERE условие_отбора;

Параметры этого запроса следующие:

- таблица — имя таблицы, в которой будут обновлены данные;

- новое_значение — выражение, которое задает новые значения полей указанной таблицы;

- условие_отбора — выражение, которое определяет, какие записи будут обновлены

Пример 3

Пусть в таблице tblTransaction есть следующие записи:

Табл. 3. Содержимое таблицы tblTransaction до обновления

ID	Date	AgentID	Amount
1	01.12.2007	1	150
2	01.12.2007	1	100
3	01.12.2007	1001	75
4	02.12.2007	1	100
5	02.12.2007	1001	100

Также пусть в таблице tblAgent есть записи, которые были добавлены инструкциями INSERT из примеров выше. Нужно увеличить сумму на 20 во всех транзакциях, в которых участвовал агент с идентификатором 1001. Это можно сделать с помощью следующего запроса:

```
UPDATE tblTransaction
    SET Amount = Amount + 20
    WHERE (AgentID = 1001)
```

Инструкция DELETE

Создает запрос на удаление записей, подпадающих под условие отбора.

Синтаксис

DELETE FROM таблица
WHERE условие_отбора

Параметры этого запроса следующие:

- таблица — имя таблицы, из которой удаляются данные;

- условие_отбора — выражение, которое задает записи для удаления, если для данной записи это условие истинно, то запись будет удалена.

Пример 4

```
DELETE FROM tblTransaction
    WHERE (ID = 2)
```

Эта инструкция удаляет из таблицы транзакций транзакцию, у которой идентификатор равен 2.

Инструкция SELECT

Эта инструкция создает запрос на выборку данных из БД. Она возвращает данные в виде набора записей. В примерах к этой инструкции будет использоваться таблица tblTransaction, данные в которой представлены в табл. 4.

Табл. 4. tblTransaction

ID	Date	AgentID	Amount
1	01.12.2007	1	150,00
3	01.12.2007	1001	95,00
4	02.12.2007	1	100,00
5	02.12.2007	1001	120,00
6	02.12.2007	1	210,00
7	02.12.2007	1001	175,00
8	02.12.2007	1	215,00
9	03.12.2007	1001	145,00
10	03.12.2007	1	140,00
11	03.12.2007	1	120,00
12	03.12.2007	1001	60,00
13	03.12.2007	1001	75,00
14	04.12.2007	1	85,00
15	04.12.2007	1	165,00
16	04.12.2007	1001	180,00
17	04.12.2007	1	215,00
18	05.12.2007	1	210,00
19	05.12.2007	1	110,00
20	05.12.2007	1001	115,00

Синтаксис

SELECT [предикат] { * | таблица.* | [таблица.]поле_1

FROM выражение [, ...]

[WHERE условие_отбора]

[GROUP BY...]

[HAVING...]

[ORDER BY...]

Параметры этого запроса следующие:

- предикат — один из следующих предикатов отбора: ALL, DISTINCT, DISTINCTROW или TOP. Предикаты используются для ограничения числа возвращаемых записей. Если они отсутствуют, по умолчанию используется предикат ALL.

- * — означает, что будут выбраны все поля заданной таблицы или таблиц,

- таблица — имя таблицы, из которой должны будут быть выбраны записи,

- поле_1 — поле из таблицы, откуда будут выбраны записи,

- выражение — имена одной или нескольких таблиц, которые содержат отбираемые данные,

- условие_отбора — выражение, задающее записи, которые попадут в результат.

Описание предикатов:

ALL — означает, что будут возвращены все записи.

DISTINCT — исключает записи, которые содержат повторяющиеся значения в выбранных полях, чтобы запись была включена в результат выполнения запроса, необходимо чтобы она отличалась от всех остальных записей.

DISTINCTROW — опускает данные, основанные на целиком повторяющихся записях, а не на отдельных повторяющихся полях.

TOP *n* [PERCENT] — указывает на то, что в результате должны быть *n* первых записей или, если указано PERCENT — *n%* первых записей запроса.

Пример 5

```
SELECT DISTINCT AgentID
    FROM tblTransaction
```

Результат выполнения этой инструкции — список идентификаторов агентов, у которых есть транзакции

Предложение GROUP BY [группируемые_поля]

Объединяет записи с одинаковыми значениями в указанном списке группируемые_поля в одну запись. Если инструкция SELECT содержит статистическую функцию SQL, например Sum или Count, то для каждой записи будет вычислено итоговое значение.

Пример 6

```
SELECT AgentID, COUNT(AgentID) AS TransactionsPerAgent
    FROM tblTransaction
    GROUP BY AgentID
```

Результат выполнения этой инструкции представлен в табл. 5 - это список идентификаторов агентов и количество транзакций с их участием. С помощью оператора AS вводится синоним для результата статистической функции (см. далее), для того, чтобы можно было обратиться к столбцу с результатом.

Табл. 5

AgentID	TransactionsPerAgent
1	11
1001	8

Предложение HAVING

Это предложение аналогично предложению WHERE, после него указывается условие, но это условие применяется уже к результату группировки после предложения GROUP BY.

Пример 7

```
SELECT AgentID, COUNT(AgentID) AS TransactionsPerAgent
    FROM tblTransaction
    GROUP BY AgentID
    HAVING  (COUNT(AgentID) > 10)
```

Этот пример аналогичен предыдущему, но выбираются идентификаторы только тех агентов, у которых количество транзакций больше 10. Результат этого запроса представлен в табл. 6.

Табл. 6

AgentID	TransactionsPerAgent
1	11

Предложение ORDER BY поле_1 [ASC | DESC][, поле_2 [ASC | DESC]][, ...]]

Сортирует записи, полученные в результате запроса, в порядке возрастания или убывания на основе значений указанного поля или полей. ASC — сортировка по возрастанию; DESC — сортировка по убыванию.

Пример 8

```
SELECT [Date], SUM(Amount) AS DayAmount
    FROM tblTransaction
    GROUP BY [Date]
    ORDER BY SUM(Amount) DESC
```

Этот запрос подсчитывает сумму по дням, сортирует результат по убыванию. Результат выполнения запроса приведен в

табл. 7. Следует отметить, что если имена столбцов или таблиц (в приведенном примере столбец Date) могут совпадать с именами зарезервированных сущностей, то их следует экранировать квадратными скобками, что, и сделано в этом примере.

Табл. 7.

Date	DayAmount
02.12.2007	820
04.12.2007	645
03.12.2007	540
05.12.2007	435
01.12.2007	245

Статистические функции

Статистические функции SQL используются для определения статистических данных на основе наборов числовых значений. Допускается использование этих функций в запросах. В некоторых примерах выше уже использовались такие функции как SUM и COUNT. Существует еще несколько статистических функций. За полным списком статистических функций следует обратиться к документации используемого диалекта языка SQL.

Выражения – аргументы функций, является строковыми выражениями, которые определяют поле, содержащее числовые данные для вычисления, или выражения, выполняющее вычисления с данными из этого поля. Операнды аргумента выражение могут включать имя поля таблицы, константу или функцию, связанные между собой знаками, например, арифметических действий.

AVG (выражение)

Вычисляет арифметическое среднее набора чисел, содержащихся в указанном поле запроса.

COUNT (выражение)

Вычисляет количество записей, возвращаемых запросом.

FIRST (выражение)

LAST (выражение)

Возвращают значение поля из первой или последней записи результирующего набора запроса.

MIN (выражение)

MAX (выражение)

Возвращают минимальное и максимальное значения из набора значений, содержащихся в указанном поле запроса.

STDEV (выражение)

STDEVP (выражение)

Возвращают смещенное и несмещенное значение стандартного отклонения, вычисляемого по набору значений, содержащихся в указанном поле запроса.

Стандартное отклонение, это параметр, который указывает величину разброса функции распределения около среднего значения. Он равен квадратному корню из момента для квадрата отклонений от среднего. Этот параметр используется для описания набора значений, чтобы определить величину отклонений значений от среднего арифметического.

VAR (выражение)

VARP (выражение)

Возвращают значение смещенной и несмещенной дисперсии, вычисляемой по набору значений, содержащихся в указанном поле запроса.

Дисперсия, это квадрат значения среднеквадратичного отклонения. Мера отличия значений в группе от среднего значения.

Объединения

Мощной возможностью инструкции DML SELECT является возможность извлекать данные из нескольких таблиц. Эта возможность реализуется с помощью объединений. Объединения, это специальные выражения, которые поме-

щаются в предложениях FROM и из результата объединения производится дальнейшая выборка данных.

Объединение INNER JOIN

Объединяет записи из двух таблиц.

Синтаксис

FROM таблица_1 INNER JOIN таблица_2 ON таблица_1.поле_1 оператор таблица_2.поле_2

Параметры этого запроса следующие:

- таблица_1 и таблица_2 — имена таблиц, подлежащих объединению;

- поле_1, поле_2 — имена полей по которым производится объединение, если эти поля не являются числовыми, то должны иметь одинаковый тип данных и содержать данные одного рода;

- оператор — любой оператор сравнения: "=", "<", ">", "<=", ">=" или "<>".

Пример 9

Модифицируем пример 6 так, что бы вместо идентификатора агента подставлялось его имя и выводилась характеристика агента, результат выполнения приведен в табл. 8.

```
SELECT tblAgent.Name,
COUNT(tblTransaction.AgentID)
AS TransactionsPerAgent, tblAgent.Notes
FROM (tblAgent INNER JOIN tblTransaction
  ON tblAgent.ID = tblTransaction.AgentID)
GROUP BY tblAgent.Name, tblAgent.Notes
```

Табл. 8.

Name	TransactionsPerAgent	Notes
Агент А	11	Отличный агент
Агент Б	8	Надежный агент

Объединение LEFT JOIN и RIGTH JOIN

Синтаксис

FROM таблица_1 [LEFT | RIGTH] JOIN таблица_2 ON таблица_1.поле_1 оператор таблица_2.поле_2

Параметры этого запроса следующие:

- таблица_1 и таблица_2 — имена таблиц, подлежащих объединению;

- поле_1, поле_2 — имена полей по которым производится объединение, если эти поля не являются числовыми, то должны иметь одинаковый тип данных и содержать данные одного рода;

- оператор — любой оператор сравнения: "=", "<", ">", "<=", ">=" или "<>".

Операция LEFT JOIN используется для создания левого внешнего объединения. Левое внешнее объединение включает все записи из первой (левой) таблицы, даже если нет совпадающих значений для записей из второй (правой) таблицы.

Операция RIGHT JOIN используется для создания правого внешнего объединения. Правое внешнее объединение включает все записи из второй (правой) таблицы, даже если нет совпадающих значений с записями из первой (левой) таблицы.

Операции объединения могут быть вложенными. Операции LEFT JOIN или RIGHT JOIN могут быть вложены в операцию INNER JOIN, но операция INNER JOIN не может быть вложена в LEFT JOIN или RIGHT JOIN.

Индексы и ограничения, накладываемые на сохраняемые данные

Индексы

Одним из основных способов достижения высокой производительности запросов к БД является использование индексов. Индекс ускоряет процесс запроса, предоставляя быстрый доступ к строкам данных в таблице, аналогично тому, как указатель в книге помогает вам быстро найти необходимую информацию.

Индексы создаются для столбцов таблиц и предоставляют путь для быстрого поиска данных на основе значений в этих столбцах. Без использования индекса будет проведен полный перебор всех строк таблицы. Индекс можно создавать для большинства столбцов таблицы. Также индекс может быть составным и содержать больше одного столбца. Индекс представляет собой иерархическую структуру, организованную в виде сбалансированного дерева. Принцип формирования и модификации сбалансированного дерева остается за рамками рассмотрения настоящего пособия. Для иллюстрации ускорения поиска по индексу рассмотрим поиск транзакций, совершенных на сумму больше определенной. Пусть для столбца `amount` таблицы `tblTransaction` задан индекс. Схематически его можно изобразить следующим образом (см. рис. 7). В значениях узлов указаны диапазоны сумм транзакций. В узлах нижнего уровня находятся ссылки на транзакции, попадающие в диапазон, указанный на узлах.

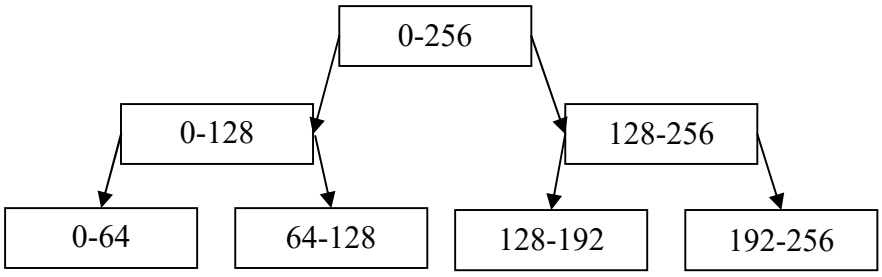


Рисунок 7. Пример структуры индекса

При формировании запроса на индексированный столбец, подсистема запросов начинает идти сверху от корневого узла и постепенно двигается вниз через промежуточные узлы. Подсистема запросов продолжает двигаться по узлам индекса до тех пор, пока не достигнет нижнего уровня с листьями индекса. Например, при выполнении следующего запроса на втором уровне будет рассматриваться только узел «128-256» и на третьем будут перебираться только записи, закрепленные за узлом «128-192». Использование бинарного поиска позволяет исключить из рассмотрения большинство записей, не подходящих под условия запроса, не анализируя каждую из записей в отдельности.

```
SELECT * FROM tblTransaction
WHERE Amount > 130 AND Amount < 180
```

Необходимо отметить, что за скорость поиска приходится «расплачиваться». Хранение большого количества индексов может занимать значительное дисковое пространство, а при добавлении новых или изменении старых данных индексы необходимо пересчитывать. Не следует создавать индексы для столбцов, по которым будет редко происходить поиск, но данные в которых будут часто изменяться.

Специальным образом созданный индекс может обеспечивать уникальность каждого значения в индексируемом столбце, такой индекс называется уникальным. Если индекс является составным, то уникальность распространяется на все столбцы индекса, но не на каждый отдельный столбец. К примеру, если создать уникальный индекс для столбцов Date и AgentID, то один агент сможет совершать только одну транзакцию в день, но при этом возможно дублирование дат, если транзакции совершены разными агентами.

Ключи

Для обеспечения целостности БД на данные, сохраняемые в таблицы, могут накладываться некоторые ограничения. Обычно в таблице есть столбец или

сочетание столбцов, содержащих значения, уникально определяющие каждую строку таблицы. Этот столбец, или столбцы, назначаются первичным ключом (Primary Key – PK) таблицы. Для первичного ключа автоматически создается уникальный индекс. В реляционных базах данных практически всегда разные таблицы логически связаны между собой. Первичные ключи используются для однозначной организации такой связи.

В нашем примере логически связанными между собой оказываются поля ID таблицы tblAgent и AgentID таблицы tblTransaction. Иллюстрация связи приведена на рисунке 8. Поле AgentID указывает на то, каким агентом из таблицы tblAgent была совершена транзакция.

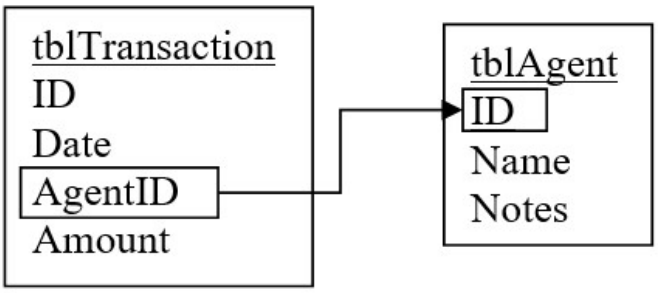


Рисунок 8. Иллюстрация связи между двумя таблицами

Предположим, что в таблице tblAgent расположены агенты с ID равным 1 и 1001. Тогда при выполнении следующего запроса возникнет нарушение целостности БД:

```
INSERT INTO tblTransaction
  (ID, Date, AgentID, Amount)
  VALUES (123, '01.01.2018', 3003, 1000)
```

Добавленная запись будет ссылаться на несуществующего агента с идентификатором 3003. Для контроля за появлением данных, которые нарушают целостность используются внешние ключи (Foreign Keys – FK). При создании внешнего ключа указываются таблица и столбцы, которые ссылаются на данные из второй таблицы, а также столбцы и таблица на которые ссылаются стро-

ки первой таблицы. При добавлении данных в первую таблицу (в которой присутствует ссылка) выполняется проверка существования строки с данными на которые совершается ссылка.

Теперь предположим, что создан внешний ключ, в котором указано что tblTransaction.AgentID ссылается на tblAgent.ID. Тогда при попытке выполнить запрос из примера выше СУБД выполнит проверку существования агента с идентификатором 3003. Проверка покажет, что такого агента не существует, запрос не будет выполнен, а в ответ на него будет сгенерирована ошибка с соответствующим описанием.

5 Технология ADO .Net

ADO.NET - это часть Microsoft .NET Framework, т.е. набор классов, позволяющих приложению легко управлять и взаимодействовать со своим файловым или серверным хранилищем данных. В .NET Framework библиотеки ADO.NET находится в пространстве имен System.Data. Эти библиотеки обеспечивают подключение к источникам данных, выполнение команд, а также хранение, обработку и выборку данных

Все необходимые классы для работы через провайдеров OLE DB или SQL находятся в пространствах имен `System.Data.OleDb` или `System.Data.SQL` соответственно.

Методы подключения к БД

Управление подключением к OLE DB или SQL источникам данных осуществляется с помощью классов `OleDbConnection` или `SQLConnection` соответственно. Самый простой способ подключения к БД – прямое указание строки подключения в конструкторе этого класса. Для формирования строки подключения в .NET 2.0 появился класс `OleDbConnectionStringBuilder`. В реальных приложениях, обычно не указывают строки подключения к базе данных в коде – гораздо эффективнее использовать для этой цели либо настройки приложения, либо отдельный файл подключения.

Для хранения параметров подключения в Windows существует специальный тип файлов Microsoft Universal Data Link – это файл с расширением udl. С этим расширением ассоциирован универсальный редактор подключений. Чтобы использовать udl-файл, выполните следующие шаги:

- Создайте пустой файл с расширением .udl;

- Откройте файл (Enter), появится связанный с данным расширением диалог для настройки подключения;

- В списке OleDb провайдеров выберете Microsoft OLE DB Provider for SQL Server;

- На закладке “Соединение” укажите:

- имя сервера;
- логин и пароль для подключения к БД;
- БД, к которой необходимо подключиться.

- Нажмите кнопку “Проверить подключение”, чтобы убедиться, что все работает, и нажмите кнопку OK для записи информации о подключении в файл.

Для использования подключения, описанного в udl-файле, достаточно явно или через `OleDbConnectionStringBuilder` задать значение свойства FileName в строку подключения, например, так:

Пример 10

```
public class DbTest
{
  private OleDbConnection connection = null;

  public void Init(string password)
  {
    OleDbConnectionStringBuilder cb =
      new OleDbConnectionStringBuilder();

    cb.FileName = AppDomain.CurrentDomain.BaseDirectory +
      @"..\..\..\sample.udl";

    cb.Add(“password”, password);

    connection =
      new OleDbConnection(cb.ConnectionString);
    connection.Open();
  }
}
```

В этом примере с помощью соответствующего свойства класса `OleDbConnectionStringBuilder` задается путь к файлу sample.udl, задается пароль для подключения к БД (не рекомендуется хранить пароль в файлах в открытом виде) и открывается подключение.

Выполнение запросов. Команды

Команды предназначены для передачи запросов базе данных. Для OLE DB и SQL провайдеров команда реализуется классами OleDbCommand и SQLCommand. Команда всегда выполняется в контексте некоторого открытого подключения к БД.

Чтобы выполнить запрос к БД, необходимо выполнить следующую последовательность действий:

- создать подключение к БД и открыть его;

- создать объект OleDbCommand или SQLCommand используя либо один из вариантов перегруженного конструктора, либо метод XXXConnection.CreateCommand().

- при необходимости задать транзакцию, в рамках которой будет выполнен запрос, используя свойство команды Transaction;

- задать текст SQL-запроса, используя свойство CommandText, если он не был задан в конструкторе;

- выполнить SQL-запрос, используя один из следующих методов: ExecuteScalar(), ExecuteReader() и ExecuteNonQuery().

Запрос, не возвращающий результатов

Для выполнения запроса, не возвращающего никаких результатов, например, запроса на добавление, удаление или обновление данных применяется метод `ExecuteNonQuery` класса `OleDbCommand`. Рассмотрим применение этого метода на следующем примере, в котором в таблицу tblAgent добавляется запись о неизвестном агенте.

Пример 11

```
public void ExecuteInsert()
{
  OleDbCommand cmd = connection.CreateCommand();

  cmd.CommandText = @"INSERT INTO tblAgent (Name, Notes)
    VALUES (?, ?)";

  OleDbParameter param_name = cmd.CreateParameter();
  param_name.Value = "Агент X";

  cmd.Parameters.Add(param_name);

  OleDbParameter param_note = cmd.CreateParameter();

  param_note.Value = "Неизвестный агент";

  cmd.Parameters.Add(param_note);

  cmd.ExecuteNonQuery();
}
```

В этом примере создается команда, в тексте команды конкретные имя и характеристика агента заменены специальными символами подстановки — знаками вопроса. Значения на их места подставляются из коллекции `Parameters` класса `OleDbCommand` по порядку появления знаков вопроса в тексте команды.

Запрос, возвращающий одно число

Если результат запроса представляет собой одно число, например, сумму, количество, среднее значение и пр., то для выполнения такого запроса следует использовать метод `ExecuteScalar` класса `OleDbCommand`. Рассмотрим применение этого метода на следующем примере, в котором рассчитывается сумма всех транзакций в таблице tblTransaction.

Пример 12

```
public void ExecuteScalar()
{
  OleDbCommand cmd = connection.CreateCommand();

  cmd.CommandText = @"SELECT SUM(Amount) AS TotalAmount
    FROM tblTransaction";

  cmd.Connection = connection;

  Console.WriteLine("Общая сумма: {0}",
    cmd.ExecuteScalar());
}
```

Результат запроса возвращается непосредственно из метода `ExecuteScalar()`.

Запрос на выборку

Если в результате запроса ожидается несколько строк данных, то для выполнения такого запроса используется метод `ExecuteReader` класса `OleDbCommand`.

Данный метод возвращает объект `OleDbDataReader`. `OleDbDataReader` используется для последовательного считывания данных. При его использовании необходимо наличие открытого подключения к базе данных.

Навигация по строкам результирующего множества осуществляется при помощи метода `Read()`, который возвращает `true`, если строка была успешно считана в локальный буфер. После этого значения полей строки можно считать посредством метода `GetValue()`. Перед первым вызовом метода `Read()` объект `OleDbDataReader` не позиционируется на первой строке результирующего множества, и для её прочтения необходимо сначала вызвать метод `Read()`.

Наиболее удобным способом чтения данных из результирующего множества является использование метода `Read()` совместно с конструкцией `while`.

Следующий пример показывает, как можно выполнить довольно сложный запрос, который возвращает отчет за определенный день (в данном случае 2 декабря 2007) по всем агентам, для каждого из них подсчитывается сумма за указанный день.

Пример 13

```
public void ExecuteReader()
{
  OleDbCommand cmd = connection.CreateCommand();

  cmd.CommandText = "SELECT tblTransaction.[Date]," +
    " tblAgent.Name," +
    " SUM(tblTransaction.Amount) AS DayAmountPerAgent" +
    "     FROM    (tblTransaction INNER JOIN tblAgent" +
    " ON tblTransaction.AgentID = tblAgent.ID)" +
    " WHERE (tblTransaction.[Date] = #12/2/2007#)"+
    " GROUP BY tblTransaction.[Date], tblAgent.Name";

  cmd.Connection = connection;

  OleDbDataReader reader = cmd.ExecuteReader();

  while (reader.Read())
  {
    Console.WriteLine(
      "Дата: {0}; Агент: {1}; Сумма: {2}",
      reader["Date"].ToString(),
      reader["Name"].ToString(),
      reader["DayAmountPerAgent"].ToString()
    );
  }
}
```

Внутри цикла `while` показано, как можно получить доступ к столбцам результата, применяя перегруженный оператор [], аргумент которого — имена столбцов, которые возвращаются в запросе.

6 Отладочные средства MS Visual Studio для работы с БД

Microsoft Visual Studio предоставляет богатые средства для разработки запросов к БД. На рис. 9 представлен общий вид этих средств. В правой части представлен “Обозреватель серверов”, в котором открыто соединение с БД. Вызывается данное окно через меню “Вид \ Другие окна \ Обозреватель серверов”. Чтобы подключиться к БД необходимо щелкнуть правой кнопкой мыши по узлу “Подключения данных” и выбрать пункт “Добавить подключение…”. Появится окно “Добавить подключение”, в котором можно изменить источник данных, указать адрес сервера с БД, параметры авторизации и протестировать соединение, когда все будет настроено верно, то нажав на кнопку OK можно получить новое соединение под узлом “Подключения данных” в окне “Обозреватель серверов”. Также с помощью окна “Обозреватель серверов” можно просмотреть, какие таблицы есть в БД, и какие колонки есть в таблицах.

Для добавления новой таблицы требуется щелкнуть правой кнопкой мыши по узлу “Таблицы” используемой БД и выбрать пункт “Добавить новую таблицу”. В появившемся окне “Конструктор” можно задать имена, тип и прочие параметры столбцов создаваемой таблицы (см. рис. 9). В правой части окна “Конструктора” можно добавлять ключи, проверочные ограничения, индексы, внешние ключи. Все изменения, сделанные в окне “Конструктора” конвертируются в SQL – запрос, который будет отправлен БД, и отображается в окне “T-SQL”. При добавлении нового элемента (таблица, внешний ключ и т.д.) значения их параметров задаются значениями по умолчанию, которые следует изменить на желаемые во вкладке “T-SQL”. Для того, чтобы совершенные изменения вступили в силу их необходимо применить нажатием кнопки “Обновить”, расположенной под заголовком окна “Конструктора”.

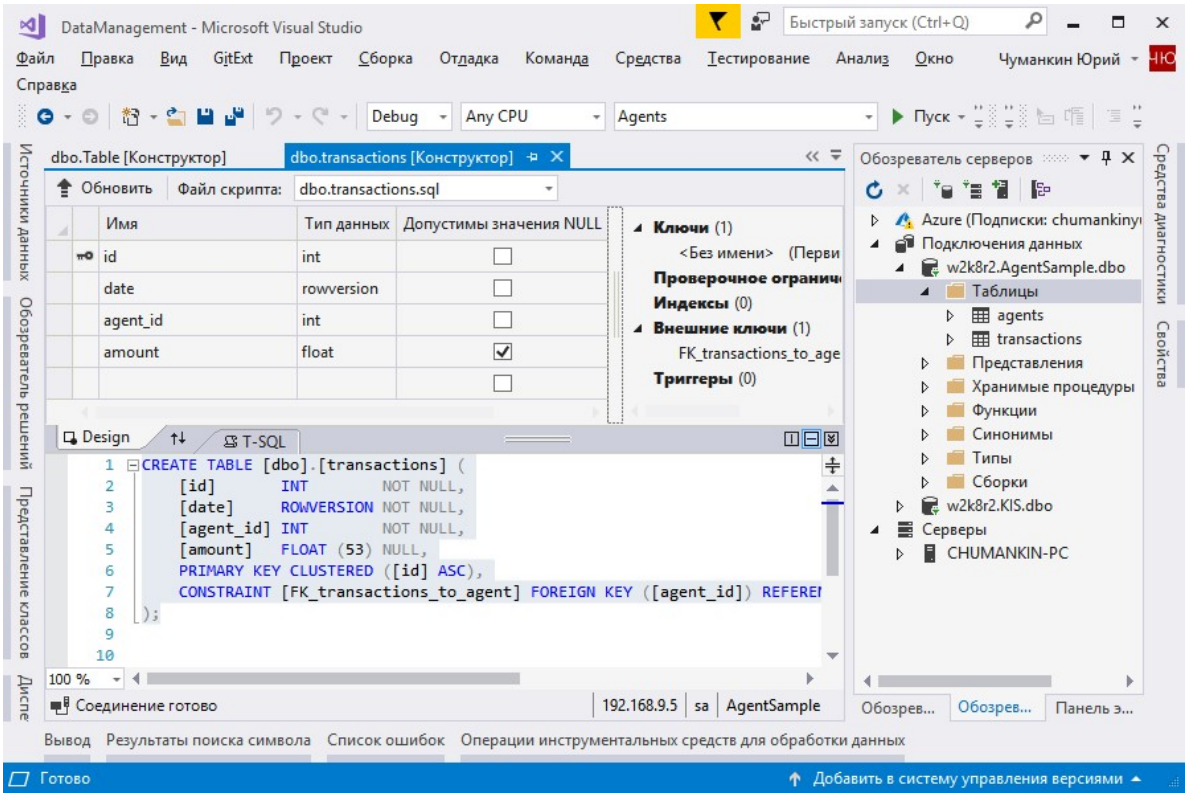


Рисунок. 9. Вид инструментальных панелей Microsoft Visual Studio для разработки запросов SQL

Для просмотра и редактирования данных, хранящихся в таблицах следует щелкнуть правой кнопкой мыши по интересующей таблице и выбрать пункт “Показать таблицу данных”.

Чтобы вызвать “Редактор запросов” — инструмент создания запросов, нужно щелкнуть правой кнопкой мыши по узлу с нужной БД и выбрать пункт “Новый запрос”. Появится окно редактирования запроса, в котором можно набрать текст запроса. В окне редактирования под заголовком расположена панель управления запросом. На которой представлены кнопки для:

- выполнения запроса;

- оценки плана выполнения (позволяет визуализировать программу выполнения запроса и оценить ресурсозатратность отдельных частей запроса, см. рис. 10);

- подключения/отключения от с сервера;

- выбора БД к которой адресуется запрос;

- выбора параметров визуализации результатов выполнения запроса.

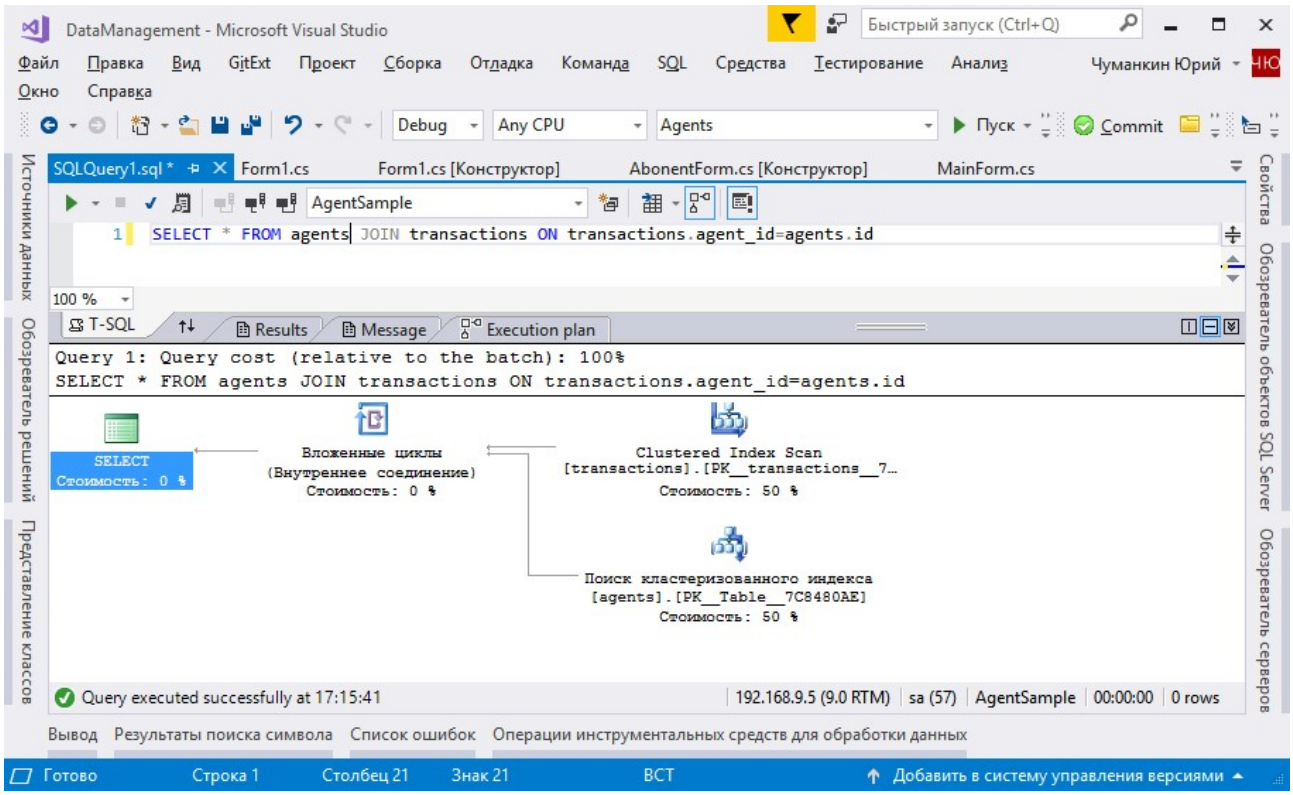


Рисунок 10. Пример отображения оценки затрат при выполнении запроса

7 Средства визуализации таблиц MS Visual Studio

Для отображения табличных данных в .NET Framework 2.0 присутствует класс `DataGridView`, обладающий богатой функциональностью и широкой поддержкой в Design-Time, множеством мастеров и редакторов комплексных свойств, облегчающих его использование. В данном разделе рассмотрим исключительно программный способ применения данного класса для визуализации результатов запроса к БД.

Пусть нам нужно получить суммы транзакций на каждый день для каждого агента отдельно, отсортированные по возрастанию даты. Это можно сделать следующим запросом.

```
SELECT tblTransaction.[Date], tblAgent.Name,
  SUM(tblTransaction.Amount) AS DayAmount, tblAgent.Notes
FROM tblAgent INNER JOIN tblTransaction
  ON tblAgent.ID = tblTransaction.AgentID)
GROUP BY tblTransaction.[Date], tblAgent.Name,
  tblAgent.Notes
ORDER BY tblTransaction.[Date]
```

Минимально необходимый код, который нужно добавить в класс формы, на которой лежит класса `DataGridView` с именем `dataGridView`, приведен ниже. Нужно проконтролировать, чтобы значение свойства `dataGridView.AutoGenerateColumns` было равно `true` (по умолчанию так оно и есть) и добавить вызов метода `Init()` в конец конструктора формы — после вызова метода `InitializeComponent()`. В этом случае элемент управления сам создаст колонки для отображения данных.

Пример 14

```
private OleDbConnection connection = null;
private BindingSource bindingSource=new BindingSource();
private OleDbDataAdapter dataAdapter = null;
private DataSet dataSet=new DataSet();

public void Init()
{
  OleDbConnectionStringBuilder cb =
    new OleDbConnectionStringBuilder();

  cb.FileName = AppDomain.CurrentDomain.BaseDirectory +
    @"..\..\..\sample.udl";

  connection = new OleDbConnection(cb.ToString());
  connection.Open();

  string selectSQL = @"SELECT tblTransaction.[Date],
      tblAgent.Name,
      SUM(tblTransaction.Amount) AS DayAmount,
      tblAgent.Notes
    FROM tblAgent INNER JOIN tblTransaction
      ON tblAgent.ID = tblTransaction.AgentID)
    GROUP BY tblTransaction.[Date], tblAgent.Name,
      tblAgent.Notes" +
    ORDER BY tblTransaction.[Date]";

  dataAdapter =
    new OleDbDataAdapter(selectSQL, connection);

  dataAdapter.Fill(dataSet,"Report");

  bindingSource.DataSource = dataSet;
  bindingSource.DataMember = "Report";

  dataGridView.DataSource = bindingSource;
}
```

В результате получим отображение результата запроса в виде таблицы (см. рис. 11).

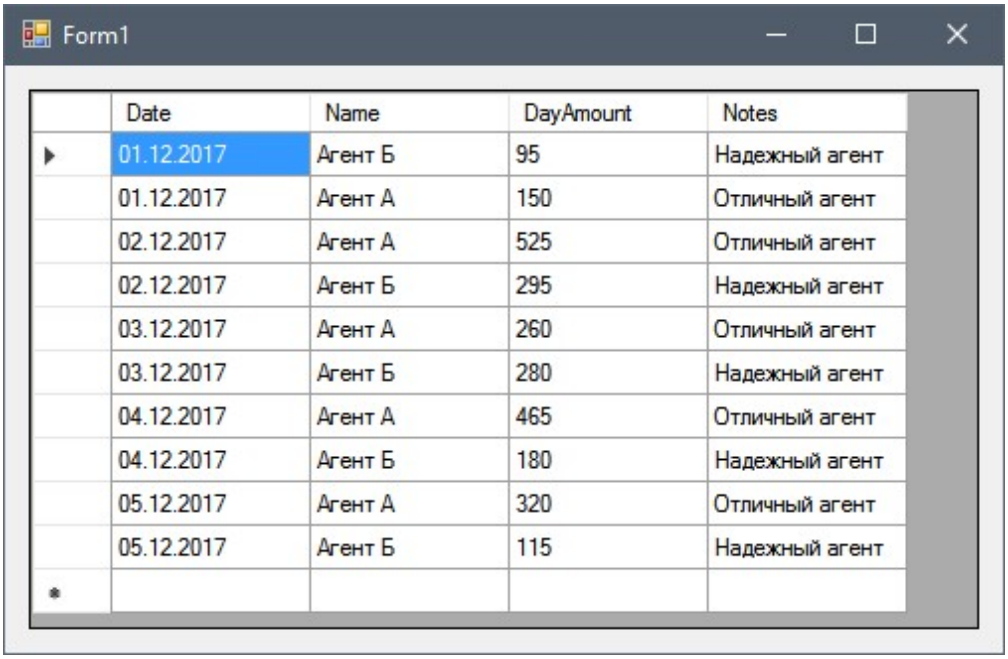
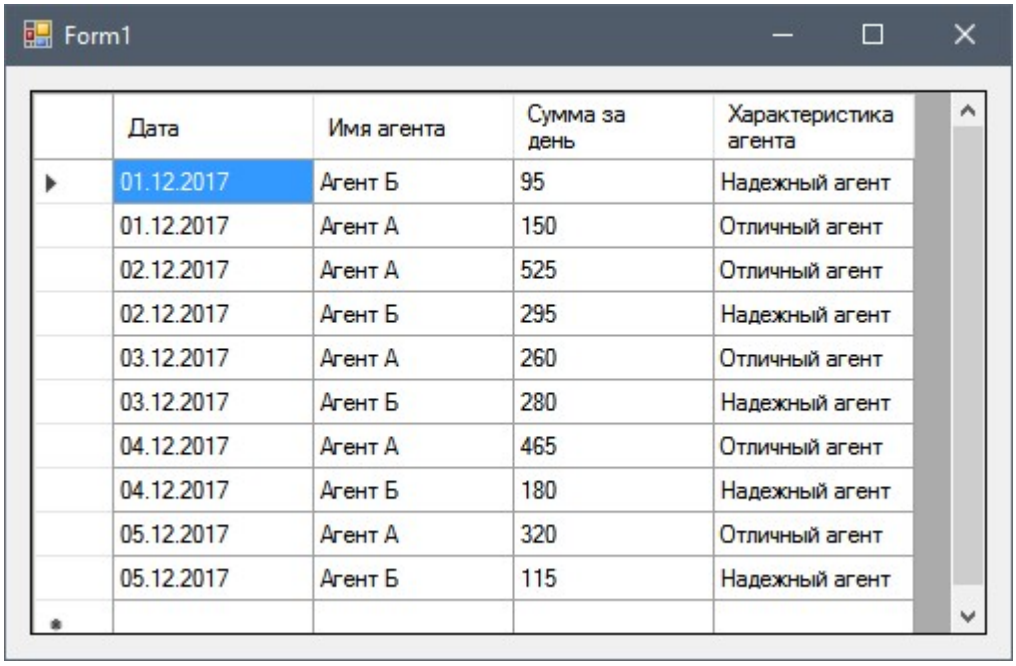


Рисунок 11. Представление результатов запроса в форме таблицы

У автоматически созданных колонок можно изменять свойства. В следующем примере показано, как можно назначить колонкам свои заголовки. Колонки находятся в коллекции Columns, доступ к ним возможен с помощью перегруженного оператора [], аргументом которого может быть номер колонки в коллекции или имя колонки. В данном случае имена колонок совпадают с именами столбцов данных в исходном запросе на выборку. Код, работающий с колонками, следует разместить после присваивания свойству `DataSource`, потому что это присваивание и приводит к тому, что создаются колонки. Результат работы модифицированного кода приведен на рис. 12.

Пример 15

```
dataGridView.Columns["Date"].HeaderText = "Дата";
dataGridView.Columns["Name"].HeaderText = "Имя агента";
dataGridView.Columns["DayAmount"].HeaderText =
  "Сумма за день";
dataGridView.Columns["Notes"].HeaderText =
  "Характеристика агента";
```



Дата	Имя агента	Сумма за день	Характеристика агента
01.12.2017	Агент Б	95	Надежный агент
01.12.2017	Агент А	150	Отличный агент
02.12.2017	Агент А	525	Отличный агент
02.12.2017	Агент Б	295	Надежный агент
03.12.2017	Агент А	260	Отличный агент
03.12.2017	Агент Б	280	Надежный агент
04.12.2017	Агент А	465	Отличный агент
04.12.2017	Агент Б	180	Надежный агент
05.12.2017	Агент А	320	Отличный агент
05.12.2017	Агент Б	115	Надежный агент

Рисунок 12. Представление результатов запроса в форме таблицы (заголовки столбцов изменены)

Следует отметить, что этот элемент управления очень большой, имеет сотни открытых свойств, методов и событий, поэтому можно рекомендовать обратится к документации в MSDN и к статьям [1, 2].

8 Учебные задания

Практические задания по курсу “Управление данными” включают в себя одно общее задание, которое выполняется и сдается каждым студентом индивидуально, и одно групповое, выполняемое и сдаваемое группами по 2 – 3 студента. Для каждой группы определяется собственное задание из заданий, приведенных в данном методическом пособии.

По окончании семестрового курса по дисциплине “Управление данными” проводится зачет. Положительная зачетная отметка “зачтено” выставляется после представления студентом и оценки преподавателем одного общего и одного индивидуального заданий.

8.1 Общее задание “Телефонный справочник”

Легенда

Требуется создать программу, облегчающую работу с множеством телефонных номеров. Условное название программы “Телефонный справочник”. Справочник относится к виду одноуровневых приложений (см. пункт 3.1). Структура данных (таблицы и отношения) создаются с помощью инструментов работы с БД среды MS Visual Studio (см. раздел 6). Приложение создается в среде MS Visual Studio на языке C# на базе платформы ADO.Net. Разрабатываемая программа должна иметь операторский интерфейс и взаимодействовать с оператором для выполнения своих функций.

Функции системы

№	Функция
1	Регистрация нового города. Регистрируются: 1. Наименование; 2. Телефонный код города; 3. Код страны.
2	Просмотр/редактирование/удаление информации о городе

3	Регистрация абонента. Регистрируются: 1. ФИО; 2. Адрес; 3. Дата рождения; 4. Текстовый комментарий.
4	Просмотр/редактирование/удаление информации о абоненте
5	Регистрация телефонного контакта. Регистрируются: 1. ФИО (выбор из зарегистрированных абонентов); 2. Наименование города (выбор из зарегистрированных городов, только для “городских” телефонов); 3. Номер телефона; 4. Тип телефона (домашний, рабочий, мобильный).
6	Просмотр/редактирование/удаление информации о телефонном контакте.
7	Поиск всех контактных телефонов для абонента (абонентов). Характеристики абонентов м. б. указаны с помощью регулярных выражений (ФИО, наименование города, адрес, дата рождения).
8	Поиск всех абонентов, доступных по указанному номеру телефона (для городских телефонов указывается наименование города).

Ограничения

Операторский интерфейс не должен отображать служебных полей таблиц БД, например, ключевые поля.

Таблицы БД должны быть идентифицированы по категориям сущностей, например, “Город”, “Абонент”, “Телефонный контакт”. Т. е. применение одной таблицы для хранения всех данных не допускается.

8.2 “Система отслеживания отчетов о проблемах” (группа 3 чел.)

Легенда

Предприятие, занимающееся сервисным обслуживанием телекоммуникационного оборудования, имеет один центр, куда обращаются клиенты, имеющие претензии к работе оборудования. На каждый запрос оформляется заявка о проблеме. Оформленная заявка просматривается с целью определения профиля проблемы и назначается ответственный за ее решение из группы мобильных специалистов. Получив заявку, ответственный за решение проблемы изменяет

статус на “Принята к исполнению” и начинает заниматься устранением проблемы (связывается с клиентом, выезжает к нему, устраняет проблему). По завершении работ мобильный специалист составляет резюме и изменяет статус запроса на “Устранена”. Группа менеджмента качества следит за быстротой реакции на запросы клиентов, скоростью устранения проблем, частотой поступления запросов и др. статистической информацией.

Жизненный цикл запроса на решение проблемы:

Время	Регистратор	Профилировщик	Мобильный специалист
1	Зарегистрирована		
2		Назначен ответственный	
3			Принята к исполнению
4			Устранена; Отложена; Не может быть решена

Функции системы

1. Регистрация

№	Функция
1	Регистрация проблемы: тип оборудования (навигация) серийный номер (навигация) тип проблемы (аппаратная проблема, программная ошибка, ошибка в программной документации, настройка) (навигация) описание (ручной ввод) приоритет (незначительная, серьезная, критическая) (навигация) источник, адрес, контактное лицо (ручной ввод) дата/время (автоматический ввод)

2	Просмотр статуса проблемы: зарегистрирована назначен ответственный принята к исполнению устранена отложена не может быть решена Просмотр даты последнего изменения статуса Просмотр резюме (текст)

2. Профилировка

№	Функция
1	Просмотр зарегистрированных запросов на устранение проблем: тип оборудования серийный номер тип проблемы описание приоритет источник дата/время регистрации
2	Назначение ответственного: выбор ФИО мобильного специалиста (навигация) изменение статуса (автоматически) изменение даты изменения статуса (автоматически) изменение приоритета
3	Регистрация мобильного специалиста: логин пароль ФИО характеристика (навыки, знания, др.)

3. Устранение проблемы

№	Функция
1	Авторизация мобильного специалиста. Ввод логина и пароля
2	Просмотр назначенных для текущего мобильного специалиста запросов на устранение проблем: тип оборудования серийный номер тип проблемы описание приоритет источник дата/время назначения
3	Изменение статуса: новый статус (принята к исполнению, устранена, отложена, не может быть решена) ввод резюме

4. Мониторинг качества обслуживания клиентов

№	Функция
1	Получение статистики за указный период: количество поступивших запросов количество устраненных проблем количество отложенных запросов количество не устраненных проблем отношение количества поступивших запросов к устраненным
2	Получение информации о наиболее активных клиентах за указанный период (10 клиентов, ранжированных по возрастанию количества запросов)
3	Получение информации о наиболее проблемном оборудовании за указанный период (10 типов, ранжированных по возрастанию количества запросов)
4	Рейтинг нагрузки мобильных специалистов за указанный период.

Ограничения

Для каждой группы функций должно быть определено собственное приложение (допускается одно приложение для всех групп функций, но с разными формами ввода/вывода и авторизацией).

8.3 “Анализатор отказов в обслуживании” (группа 3 чел.)

Легенда

Оператор спутниковой связи предоставляет абонентам услуги связи. Всего в распоряжении оператора *N* каналов связи. Один абонент полностью занимает один канал при передаче своих данных. Абонент может занимать канал на произвольное время из интервала *T0 – T* мин. Канал освобождается по инициативе абонента. Если канал не предоставляется абоненту в течении *K* сек, абонент получает извещение об отказе в обслуживании. Необходимо создать программную систему, моделирующую использование ограниченного количества каналов несколькими абонентами и рассчитывающую статистические характеристики загрузки системы в зависимости от числа каналов, числа абонентов, длительности таймаута ожидания канала, среднего и максимального времени работы абонента. Назначение системы – определение оптимального количества каналов (аренда канала достаточно дорога) при заданном уровне качества сервиса.

Функции системы

1. Модель активности абонентов

№	Функция
1	Регистрация абонентов, редактирование, удаление. Фиксируются: идентификатор (наименование) качество обслуживание (высокое, среднее, низкое)
2	Управление характеристиками модели. Задаются/редактируются: число каналов *N* минимально и максимальное время работы абонента в эфире *T0 - T* мин таймаут ожидания канала *K* (для трех уровней качества обслуживания), с

3	Эмуляция выхода абонентов в эфир. Все зарегистрированные абоненты циклически (время сеанса случайное *T0 – T* мин) генерируют запросы на обслуживание. Фиксируются: идентификатор время поступления запроса Если запрос удовлетворен (канал выделен), то абонент занимает канал на случайный интервал в диапазоне *T0 – T* мин, по истечении интервала абонент уведомляет о завершении сеанса. Запросы на обслуживание от конкретного абонента не генерируются, пока он не освободит канал.

2. Планировщик сеансов

№	Функция
1	Обработка запросов абонентов, распределение каналов. С интервалом 0,5 сек сканируется очередь запросов абонентов на обслуживание. Обнаружив в очереди абонента, проверяется наличие свободных каналов и если такие есть, то каналы предоставляются абонентам, обнаружив уведомление о завершении сеанса, из очереди запросов удаляются запрос на обслуживание и соответствующее уведомление о завершении. В случае отсутствия свободных каналов запросы на обслуживание удаляются из очереди через К(качество обслуживания) сек. (событие “отказ в обслуживании”). Фиксируются: идентификатор абонента запрашивающего обслуживание время поступления запроса на обслуживание время поступления уведомления о завершении сеанса номер канала сеанса

3. Анализатор

№	Функция
1	Получение статистической информации за указанный период: общее количество отказов в обслуживании суммарное время “простоя каналов” средняя длительность сеансов максимальная длительность сеансов минимальная длительность сеансов

Ограничения

Для каждой группы функций должно быть определено собственное приложение (допускается одно приложение для всех групп функций, но с разными формами ввода/вывода и авторизацией).

8.4 “Биллинговая система” (группа 2 чел.)

Легенда

Оператор спутниковой связи предоставляет абонентам услуги связи. Плата за услуги пропорциональна времени работы абонента в эфире и зависит от тарифного плана. Необходимо создать программную систему, собирающую первичную информацию и формирующую счет для каждого абонента.

Функции системы

1. Модель

№	Функция
1	Регистрация абонентов, редактирование, удаление. Фиксируются: идентификатор (наименование) тарифный план начало обслуживания период действия договора
2	Эмуляция выхода абонентов в эфир. Все зарегистрированные абоненты независимо (время сеанса случайное 0 – 1 мин) выходят в эфир. Фиксируются: идентификатор время начала сеанса время завершения сеанса текущий тарифный план тип передаваемых данных (голос, пакетные данные, RT-видео) количество переданных байт

2. Расчетный отдел

№	Функция
1	Выписка счета. Указываются: идентификатор абонента (навигация, поиск) период (начало периода, один из периодов: 1 мес; 4 мес; 6 мес; 12 мес.) Предоставляется счет: идентификатор абонента период (начало, завершение) сумма

Ограничения

Для каждой группы функций должно быть определено собственное приложение (допускается одно приложение для всех групп функций, но с разными формами ввода/вывода и авторизацией).

8.5 “Иерархическая записная книжка” (группа 2 чел.)

Легенда

Иерархическая записная книжка, это приложение, которое позволяет вести записи и делать заметки, организуя их в древовидную структуру.

В левой половине главного окна пользователь может редактировать иерархические взаимосвязи между записями, т.е. перемещать их от одного родителя к другому, удалять, создавать новые.

В правой половине окна отображается содержимое текущей заметки. А именно:

- краткое наименование (совпадает с текстом элемента дерева слева);

- описание (обычный текст или текст в формате RTF);

- список присоединенных картинок с возможностью добавлять и удалять,

- поле для просмотра картинок или их уменьшенных копий (preview и возможность просмотреть каждую картинку, возможно в отдельном окне);

- список гиперссылок с возможностями добавления, редактирования, удаления и запуска браузера для навигации по ним

Возможен поиск по названию заметки, тексту, именам файлов-картинок и адресам в гиперссылках.

Прообраз — MyBase (WjjSoft.com).

Функции системы

№	Функция
1	Добавление новой записи
2	Просмотр существующей записи: текст заметки список картинок (preview) сами картинки в натуральную величину в отдельном окне список гиперссылок и открытие их в браузере
3	Навигация по записям
4	Редактирование существующей записи: краткого наименования (того, что отображается в дереве) непосредственно текста заметки удаление и добавление новых картинок удаление, изменение и добавление гиперссылок
5	Удаление записи
6	Поиск по следующим атрибутам записи: краткое наименование текст заметки имена файлов картинок адреса сайтов в гиперссылках

8.6 “Информационная служба оптовой фирмы” (группа 3 чел.)

Легенда

Оптовая фирма торгует продуктами питания. Ассортимент товаров включает как фасованные продукты (лимонад, шоколад, печенье, др.), так и весовые (сахар-песок, картофель, морковь, др.). Фирма состоит из четырех подразделений: склад, отдел продаж, бухгалтерия, отдел маркетинга. Подразделения разнесены территориально. Необходимо создать распределенную программную систему, обеспечивающую совместную работу всех четырех подразделений.

Функции системы

1. Склад

№	Функция
1	Регистрация товара, поступающего на склад. Фиксируется: наименование товара количество единицы измерения дата поступления дата истечения срока хранения температура хранения цена закупки размещение (на складе)
2	Поиск товаров с истекшим сроком хранения. Предоставляется информация: наименование товара количество единицы измерения дата истечения срока хранения размещение (на складе) Составляется акт списания.
3	Выдача товара (по накладной или по акту списания). Фиксируется: номер документа тип документа дата Предоставляется: наименование товара количество единицы измерения размещение (на складе)

2. Бухгалтерия

№	Функция
1	Назначение продажной цены товара. Указывается товар (навигация, поиск), устанавливается цена.
2	Расчет прибыли за период (по заказам)

3. Отдел продаж

№	Функция
1	Оформление заказа. Фиксируется товар из списка доступных (в одном заказе м. б. несколько товаров), количество (д. б. не более чем есть на складе), клиент, дата заказа (авто). Предоставляется: сумма. Накладная с запрошенной и рассчитанной информацией сохраняется в текстовом файле. Возможность отмены заказа на любом этапе.

4. Отдел маркетинга

№	Функция
1	Определение 10-ти наиболее часто заказываемых товаров (за определенный период)
2	Определение 10-ти наименее часто заказываемых товаров (за определенных период)
3	Определение 10-ти наименее прибыльных товаров (за определенный период)
4	Определение 10-ти наиболее прибыльных клиентов

Ограничения

Для каждой группы функций должно быть определено собственное приложение (допускается одно приложение для всех групп функций, но с разными формами ввода/вывода и авторизацией).

8.7 “Метрологическая служба” (группа 1 чел.)

Легенда

Крупная организация использует для своих нужд большое количество контрольно-измерительного оборудования. Согласно закону РФ “О единстве измерений” каждый прибор должен быть отнесен к одному из утвержденных типов, для него должны быть определены: процедура поверки, межповерочный интервал, организация – поверитель, план поверки. На предприятии организо-

вана специальная служба – метрологическая, которая занимается регистрацией всех применяемых приборов, слежением за своевременным выполнением поверок аппаратуры и др. связанными работами. Необходимо разработать программу, автоматизирующую учет приборов, слежение за их статусом и составление планов поверок.

Функции

№	Функция
1	Регистрация нового прибора: тип наименование ответственный размещение серийный номер интервал поверки дата последней поверки
2	Удаление информации о приборе
3	Получение перечня приборов нуждающихся в поверке (за две недели до истечения срока поверки)
4	Получение информации о конкретном приборе (поиск)
5	Просмотр/навигация по приборам (информация о приборе + статус)
6	Отправка приборов на поверку: серийный номер (навигация) когда отправлен (ручной ввод) когда получен (ручной ввод) поверитель (навигация)
7	Ведение списка организаций – поверителей (название организации, адрес, типы поверяемых приборов).

8.8 “Мониторинг процессов” (группа 1 чел.)

Легенда

Служба тестирования программного обеспечения фирмы “R & K Software” обратилась в службу инструментальных средств с запросом на разработку ПО, обеспечивающего сбор статистической информации о работе поль-

зователей с приложениями, имеющими графический интерфейс. Описание требуемой функциональности приведено ниже.

Функции системы

№	Функция
1	Слежение за программами с операторским интерфейсом 1. При запуске новой программы фиксируется информация (наименование, время запуска); 2. При завершении программы фиксируется информация (наименование, время завершения).
2	Просмотр статистических характеристик: имя программы, общее время работы, количество запусков, средний интервал между запусками, посл. дата/время запуска. Фильтры: по временному интервалу по наименованию

Дополнительная информация

Пример получения заголовка окна.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;

namespace AppView
{
  class Program
  {
    static void Main(string[] args)
    {
      //Получаем список процессов
      Process[] procs = Process.GetProcesses();
      //Итерируемся по //всему списку процессов
      foreach (Process proc in procs)
      {
        //Если у процесса заголовок окна не равен "",
        //печатаем его на экран.
        if (proc.MainWindowTitle != "")
        {
          Cosole.WriteLine(proc.MainModule.FileName);
        }
      }
    }
  }
}
```

8.9 “Учет средств коммуникации и вычислений” (группа 2 чел.)

Легенда

Крупная научно-производственная организация использует большое количество вычислительной техники (компьютеров) и средств коммуникации (маршрутизаторы, коммутаторы и др.). Парк оборудования постоянно обновляется. Закупаются как полные комплекты (например, системный блок, монитор, клавиатура и мышь), так и комплектующие (например, видео платы, сетевые платы, платы ввода/вывода). Каждая единица оборудования ставится на баланс

предприятия, назначается ответственный, определяется размещение, фиксируются идентификационные, функциональные и коммуникационные параметры. Необходимо разработать программное средство, позволяющее производить учет имеющейся аппаратуры.

Функции системы

1. Материально-технический отдел

№	Функция
1	Регистрация компьютера. Фиксируется: наименование серийный номер дата истечения гарантийного срока продавец тип дата поступления цена размещение ответственный
2	Навигация, редактирование/удаление компьютеров: размещение ответственный
3	Регистрация комплектующих Фиксируется: наименование серийный номер дата истечения гарантии продавец тип дата поступления цена размещение (склад, компьютер) ответственный
4	Навигация, редактирование/удаление комплект: размещение ответственный
5	Поиск материальных ценностей по: ответственному размещению типу наименованию

2. Служба эксплуатации

№	Функция
1	Навигация, редактирование компьютеров: размещение ответственный сетевое имя IP маска подсети …..входящие в состав компьютера комплектующие
	Поиск мат. цен. по: ответственному размещению типу наименованию сетевому имени (для компьютеров) IP (для компьютеров) маске подсети (для компьютеров) серийному номеру

Ограничения

Для каждой группы функций должно быть определено собственное приложение (допускается одно приложение для всех групп функций, но с разными формами ввода/вывода и авторизацией).

8.10 “Учет отработанного времени сотрудников” (группа 3 чел.)

Легенда

Строительная площадка имеет многотысячный штат сотрудников. Сотрудники делятся на бригады и могут работать на различных объектах на площадке. Вход и выход на каждый объект осуществляется через КПП с турникетами. Проходы сотрудников через турникет фиксируются.

Руководство стройки должно получать отчеты:

- о количестве человек, работавших на объекте за произвольный период;

- о количестве человеко-часов, отработанных определенной бригадой;

- о количестве человеко-часов отработанных на объектах.

Необходимо автоматизировать построение отчетов.

Функции системы

1. Отдел кадров

№	Функция
1	Регистрация сотрудника. Фиксируется: ФИО табельный номер дата приёма на работу номер бригады должность
2	Редактирование/удаление сотрудников: ФИО табельный номер дата приёма на работу номер бригады должность
3	Поиск сотрудников по: ФИО табельному номеру дате приёма на работу номеру бригады должности

2. Отдел обслуживания КПП

№	Функция
1	Регистрация проходов через турникет: сотрудник; время прохода; цех; совершен вход или выход.
2	Просмотр проходов через турникет с применением фильтров по: сотруднику; времени прохода; цеху; совершенному входу или выходу.

3. Отдел формирования отчетов

№	Функция
1	Формирование и просмотр отчетов по работавшему количеству человек с возможностью выбора: объекта бригады специальности
2	Формирование и просмотр отчетов по отработанным человеко-часам с возможностью выбора: объекта бригады специальности

Ограничения

Для каждой группы функций должно быть определено собственное приложение (допускается одно приложение для всех групп функций, но с разными формами ввода/вывода и авторизацией).

9 Рекомендуемая литература

1 Н. Щербунов «DataGridView. Новый контрол в составе Framework 2.0. Обзор архитектуры, свойств и возможностей. Практика использования» // RSDN Magazine #1-2006 (http://gzip.rsdn.ru/article/dotnet/DataGridView20.xml)

2 Н. Щербунов «DataGridView. Новый контрол в составе Framework 2.0. Часть 2. Обзор архитектуры, свойств и возможностей. Практика использования.» // RSDN Magazine #2-2006 (http://gzip.rsdn.ru/article/dotnet/DataGridView20part2.xml)

3 С. Малик Microsoft ADO.NET 2.0 для профессионалов. : Пер. с англ. — М.: ООО «И.Д. Вильямс», 2006. — 560 с.: ил.

4 Л. Бейли Изучаем SQL Пер. с англ. — С.-П.: «Питер», 2012. — 582 с.: ил.

5 Дж. Боуман, С. Эмерсон, М. Дарновски Практическое руководство по SQL. : Пер. с англ. — М.: ООО «И.Д. Вильямс», 2002. — 318 с.: ил.

Сергей Алексеевич **Минеев**
Юрий Евгеньевич **Чуманкин**

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММ, ВЗАИМОДЕЙСТВУЮЩИХ С БАЗАМИ ДАННЫХ

Учебно-методическое пособие

Государственное образовательное учреждение высшего профессионального образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»
603950, Нижний Новгород, пр. Гагарина, 23.

Издание в электронной форме