Московский государственный технический университет
гражданской авиации
Факультет прикладной математики и вычислительной техники
Кафедра прикладной математики

Коновалов В.М.

Прикладное программное обеспечение

ПОСОБИЕ

к выполнению лабораторных работ

часть II

для студентов IV курса
специальности "Прикладная математика"
дневного обучения

Москва – 2011

Данное пособие издается в соответствии с рабочей программой учебной дисциплины «Прикладное программное обеспечение» по Учебному плану для студентов IV курса специальности "Прикладная математика" дневного обучения, утвержденному в 2007 г. Рассмотрено и рекомендовано к изданию на заседании кафедры и методического совета 24. 12. 2010 г.

СОДЕРЖАНИЕ

Предисловие

В пособии рассматриваются объектные модели архитектуры Universal Data Access (UDA) – стратегии доступа к данным, которую Microsoft реализует в семействе своих операционных систем. С точки зрения разработчика ***прикладного*** программного обеспечения, интерес представляет единообразный (универсальный) способ работы с разнообразными источниками данных. Высокоуровневые интерфейсы, входящие в состав UDA, предоставляют эту возможность разработчикам приложений.

Структура содержания и изложение материала данного пособия подчинены целям ***обучения***. Вначале приводятся общие сведения о наиболее известных методах доступа к источникам данных. Затем следует более детальное рассмотрение технологий, на базе которых студенты выполняют лабораторный практикум.

Приведенные в пособии последовательность выполнения лабораторных работ и объем заданий на их выполнение определяют ***минимум*** для прохождения практикума. Вместе с тем, нет никаких ограничений на самостоятельный выбор студентом стратегии использования компонентов UDA. При этом следует учитывать то обстоятельство, что реализуемая функциональность методов доступа определяется, в том числе, техническими возможностями и временными ограничениями на выполнение лабораторных работ.

Основная задача лабораторного практикума состоит в получении временных оценок методов доступа с целью выявления их относительных преимуществ и выдачи рекомендаций по использованию.

Понятно, что результаты тестирования являются достаточно условными. И абсолютные, и относительные характеристики могут существенно меняться в зависимости от конкретных условий: оборудования, системного ПО, квалификации разработчика и, конечно, структуры и содержимого базы данных. Но подобные оценки представляют интерес хотя бы потому, что заставляют разработчика внимательнее изучить возможности повышения эффективности своих приложений, в том числе проведя соответствующее исследование разных методов.

Технологии доступа к базам данных постоянно развиваются. Пока осваивается одна технология, уже появляется другая. Только одно остается неизменным: доступ к внешним источникам данных играет все более важную роль при разработке приложений. Знание технологий и эволюционных изменений, которые они вызывают, поможет найти оптимальную технологию для текущей задачи и сделать обоснованный выбор в случае необходимых изменений.

Информационные технологии прямого доступа к данным.

В современных условиях развития средств разработки и эксплуатации информационных систем (ИС) проектировщику или программисту ИС приходится решать **задачу выбора методов и средств доступа к источникам внешних данных**, иначе - задачу выбора технологии доступа к БД. Это - одна из стратегических задач, от решения которой зависит как производительность системы и способность её к реализации дополнительной функциональности, так и совместимость с другими программными платформами и технологиями.

Существует несколько способов решения задачи обеспечения доступа к данным, в процессе которого приходится разбираться во всем многообразии технологий: ODBC, DAO, RDO, OLE DB, ADO и RDS, BDE, ADO**.**NET. Изучение технологий доступа к данным **в контексте их эволюции** упрощает проблему выбора, поскольку появляется возможность сравнительного анализа и последующей оптимизации выбранной технологии для своих целей.

Как отмечено выше, взаимодействие приложения с БД может быть организовано различными способами. Первоначально преобладали способы, **основанные на API** (Application Programming Interface) библиотек СУБД (позже, дополнительно, и COM объектов), входящих в состав клиентского программного обеспечения, устанавливаемого на компьютерах пользователей. Как правило, клиентское программное обеспечение включало в себя и собственную среду разработки прикладного программного обеспечения. Это приводило к тому, что замена СУБД требовала значительной переделки кода приложения. Как результат - в настоящее время востребованными становятся **универсальные** механизмы доступа к данным, обеспечивающие для клиентского приложения стандартный набор общих функций, классов, сервисов, необходимых для работы с различными СУБД.

Универсальные механизмы доступа к данным обычно реализованы в виде **библиотек** и **дополнительных** модулей (драйверов или провайдеров). **Библиотеки** содержат определенный **стандартный** набор классов, методов, параметров, и обеспечивают стандартный интерфейс доступа к данным. **Дополнительные модули** реализуют непосредственное обращение к функциям API конкретных СУБД. Причем, эти дополнительные модули, устанавливаются исходя из текущей потребности, а изменения кода приложения не требуется.

Вместе с тем, универсальные методы не лишены недостатков (снижение быстродействия, поставка соответствующих драйверов и их настройка), но все это не идет в сравнение с их основным достоинством - универсальностью. Нельзя забывать и о том, что они значительно увеличивают производительность труда программиста, так как отпадает

необходимость в изучении специфических интерфейсов и специфических средств разработки.
Среди универсальных методов доступа к данным наиболее распространены (http://wladm.narod.ru/C_Sharp/bdbegin.html):

- ODBC - **O**pen **D**atabase **C**onnectivity.
- OLE DB - **O**bject **L**inking and **E**mbedding **D**ata**B**ase.
- ADO - **A**ctiveX **D**ata **O**bjects.
- BDE - **B**orland **D**atabase **E**ngine.
- ADO.NET - **A**ctiveX **D**ata **O**bjects for **.NET**.

ODBC (Microsoft) - технология взаимодействия **с реляционными** базами данных. Для обеспечения доступа к БД необходимы клиентская часть СУБД, ODBC-драйвер для доступа к этой СУБД (или только драйвер для некоторых СУБД) и соответствующая настройка ODBC на компьютере (обычно имя драйвера, имя пользователя, имя базы, пароль доступа и указание некоторых параметров драйвера). Драйвер, при выполнении приложения, загружается в адресное пространство приложения и используется для доступа к БД. Для каждой СУБД используется собственный ODBC-драйвер. ODBC API стандартизован. В визуальных средах разработки теперь достаточно помещения на форму соответствующего компонента, указания источника данных ODBC, имени таблицы, и связь с базой данных установлена или через клиентскую часть СУБД или через соответствующий драйвер.
OLE DB (Microsoft) - технология работы **с разнообразными** источниками данных (в отличии от ODBC - где работа производится только с реляционными БД) на основе COM-интерфейса (**C**omponent **O**bject **M**odel). OLE DB определяет набор COM-интерфейсов, включающих различные сервисы однотипного доступа к различным данным (в том числе и к нереляционным БД, например, к папкам систем электронной почты или папкам файловой системы), обеспечивая при этом поддержку работы с наборами данных и иерархическими наборами записей, подключенными непостоянно к сети.
Для доступа к БД требуется установка OLE DB провайдера для СУБД (DLL СУБД загружается при выполнении приложения в его адресное пространство). Кроме того, фирмой разработан специальный провайдер (Microsoft OLE DB Provider for ODBC Drivers), который может работать не через API клиентской части СУБД, а через интерфейс ODBC API.
ADO (Microsoft) - своеобразная **надстройка над OLE DB** (использует библиотеки OLE DB), представляющая собой дополнительный набор библиотек, содержащих COM-объекты, реализующие интерфейс доступа к данным. Этот набор библиотек **первоначально** включал две объектные технологии Microsoft: **D**ata **A**ccess **O**bjects (DAO) и **R**emote **D**ata **O**bjects (RDO) - два различных механизма доступа к локальным и удаленным базам данных, соответственно. Как ответ на потребность создания технологии,

обеспечивающей единый подход при работе с различными БД и единый интерфейс для доступа к локальным и удаленными данным, появилась технология ADO. ADO является более дружественной оболочкой базовой технологии OLE DB и позволяет работать с любыми базами данных.
BDE (Borland) - универсальный механизм доступа к данным, базирующийся на двух группах библиотек-драйверов (SQL Links - для серверных СУБД и ODBS Links - для серверных и автономных СУБД). BDE поддержан на уровне компонент в визуальных средах разработки фирмы Borland. Реализация механизма позволяет приложению обращаться к функциям клиентского API конкретной СУБД, или через ODBC API. Для доступа к базе данных с помощью BDE на компьютере должны быть установлены библиотеки BDE общего назначения (обычно устанавливаются вместе со средой разработки), а также BDE-драйвер для данной СУБД.
ADO.NET (Microsoft) - технология работы с базами данных в three-tier (многоярусной архитектуре), когда соединение с базой данных устанавливается лишь на период выполнения операций с БД (как правило кратковременных). Ее появление связано с необходимостью разрешения противоречия между ростом числа обращений к БД (особенно в БД интернет-серверов) и невозможностью базы данных поддерживать неограниченное число активных соединений. ADO.NET призвана решить эти и другие проблемы и вместе с тем сохранить удобство и простоту программирования. ADO.NET, в ее современном виде - иерархический набор объектов, построенный в соответствии с новой идеологией базовых библиотек классов и смены протоколов COM на **.**NET.

Архитектура Universal Data Access

Перечисленные выше универсальные методы доступа к внешним данным интегрированы в модели **U**niversal **D**ata **A**ccess (UDA). В соответствии с утверждением Microsoft: архитектура UDA представляет собой концепцию обеспечения доступа ко всем типам данных в масштабах предприятия. Она обеспечивает высокопроизводительный доступ к различным информационным источникам (включая как реляционные, так и нереляционные), в том числе к данным, хранящимся на мэйнфреймах, данным электронной почты и файловой системы, текстовым, графическим и географическим данным и др.
Для многих современных приложений, использующих данные, характерно подобное разнообразие их источников. Более того, могут появляться новые форматы данных и способы их хранения, поэтому разумным требованием к универсальному механизму доступа к данным является возможность поддержки не только существующих в настоящее время форматов и источников данных, но и форматов данных, которые будут созданы в будущем.

Microsoft Universal Data Access (UDA)-это архитектура фирмы Microsoft, обеспечивающая доступ к реляционным и нереляционным данным в различных форматах, которые могут храниться на различных платформах. UDA предоставляет простой программный интерфейс, использование которого возможно практически из любого современного средства разработки приложений. Чтобы освоить новую архитектуру, совсем необязательно менять собственно средство разработки, будь то Visual Basic, C++, Delphi или Microsoft Office.

Microsoft UDA базируется на открытых стандартах и не требует использования технологий, предоставляемых только одним производителем. В настоящее время Microsoft UDA поддерживается всеми основными базами данных. Одним из достоинств данной архитектуры является то, что для ее использования не требуется переноса существующих данных в единое хранилище. Напротив, Microsoft UDA поддерживает все существующие источники данных и даже облегчает их использование, предоставляя единый, универсальный способ доступа к данным.

Прежде чем обратиться к более детальному рассмотрению Microsoft UDA, следует отметить, что данная архитектура является одной из составных частей стратегии Microsoft Windows **D**istributed Inter**N**et **A**pplications (Windows DNA). Для разработчиков архитектура Microsoft UDA доступна через компоненты **M**icrosoft **D**ata **A**ccess **C**omponents (MDAC), которые включают в себя **A**ctiveX **D**ata **O**bjects (ADO), **R**emote **D**ata **S**ervices (RDS), OLE DB и ODBC. Именно эти компоненты лежат в основе Microsoft UDA и делают данную архитектуру работоспособной.

Набор библиотек MDAC является составной частью ОС Windows, начиная с Windows 2000, а для пользователей других платформ доступен отдельно на Web-сайте Microsoft.

Существующие способы доступа к данным, обычно в той или иной степени используют перечисленные универсальные механизмы или непосредственно клиентские API. С целью адаптации дальнейшего изложения материала к учебному характеру изучаемой дисциплины, рассмотрим **одну из возможных** схем такого использования компонентов UDA, представленную на рис.1.

По сути, данная схема отражает эволюцию способов доступа к данным - от создания средств **низкоуровневого** (клиентский API, ODBC, OLE DB) доступа к данным, требующего от программиста глубокого понимания структуры хранимой информации и возможностей ее извлечения и записи, а также необходимости написания достаточно объемного программного кода, до создания **высокоуровневых** (DAO, ADO) способов доступа, призванных упростить процедуру программирования операций извлечения данных из структур, в которых они хранятся в БД, и записи изменений в эти структуры.

Как видно из приведенной схемы, приложение, использующее внешние данные, может применять различные стратегии использования компонентов UDA для реализации доступа к данным. Например:

- Вызов функций ODBC API.

- Вызов функций JET API.
- Непосредственное обращение к интерфейсам OLE DB.
- Применение DAO.
- Применение ADO.
- Применение ADO + OLE DB + ODBC.

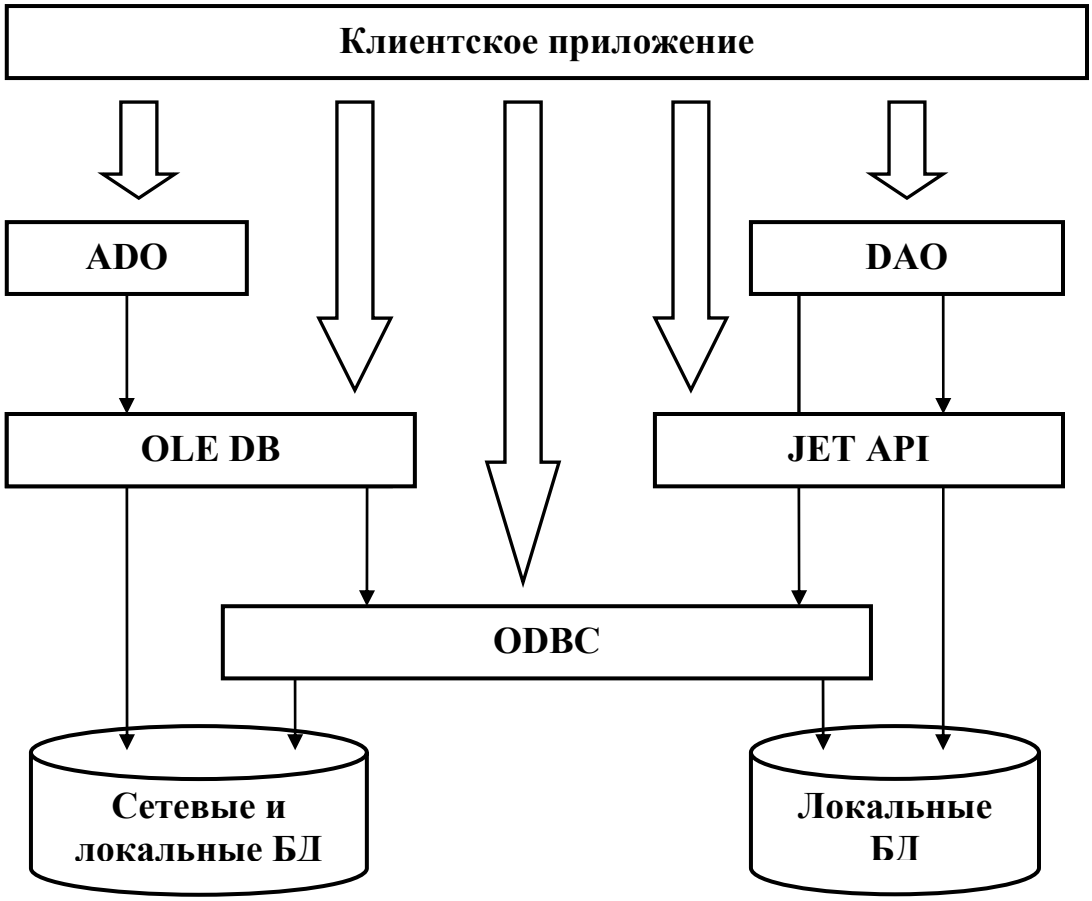

Рис.1 Стратегии использования компонентов UDA.

Перейдем к более детальному рассмотрению аспектов использования компонентов рассматриваемой архитектуры UDA, с целью выбора наиболее приемлемой технологии доступа к внешним данным в конкретных случаях.

Эволюция технологий доступа к данным

Большинство наиболее популярных и практически используемых технологий доступа к данным предоставляется фирмой Microsoft. Раньше в Microsoft ключевыми технологиями доступа к данным считались Data Access Objects (DAO) для настольных систем и Remote Data Objects (RDO), основанная на ODBC, – для клиент-серверных баз данных.

Data Access Objects (DAO).

Data Access Objects (DAO) – «родной» интерфейс программирования процессора базы данных Microsoft Jet, первоначально создавался для инструментальных сред разработки приложений Visual Basic и Visual Basic for Applications (VBA).

Использование DAO эффективно для доступа к локальным источникам данных типа Microsoft Access, Microsoft FoxPro и др., хотя сама технология вполне пригодна для доступа к удаленным источникам. Кроме всего прочего, DAO обеспечивает объектно-ориентированный интерфейс, предназначенный для выполнения всех функций, которые связаны с базой данных.

DAO – это не только чрезвычайно мощная, но также достаточно удобная технология, имеющая следующие функциональные возможности.

- Создание и редактирование баз данных, таблиц, запросов, индексов, полей, правил целостности на уровне ссылок и защиты.
- Возможности обращения к данным с помощью SQL, а также методы, полезные для управления и поиска данных в таблицах и данных, следующих из запросов.
- Поддержка транзакций, их начала, совершения и отмены. Транзакции могут быть представлены в форме вложений и являются эффективным средством для выполнения большого количества операций с базой данных как единого действия.
- Восстановление и сжатие базы данных с помощью языка программирования.
- Поддержка присоединения к базе данных разработчика удаленных таблиц, а также управление соединением.

Если говорить об Access, то здесь **процессором баз данных** является такой продукт Microsoft, как Jet (**MS Jet**). Именно этот компонент всесторонне обрабатывает базу данных. А приложение Access – просто графическая оболочка, с помощью которой пользователь взаимодействует с Jet.

Поскольку Jet – это отдельный компонент, другие прикладные программы тоже могут его использовать. Чтобы сделать взаимодействие базы данных с Jet более простым, в помощь разработчику предлагается модель программирования DAO, представляющая собой интерфейс автоматизации (**COM интерфейс Automation)** для доступа к компоненту Jet. Это означает, что любая прикладная программа, которая поддерживает **Automation**, включая все поддерживающие язык программирования VBA приложения Microsoft, может отдельно обращаться к DAO и, следовательно, к процессору базы данных. Таким образом, DAO – это объектно-ориентированная модель программирования.

DAO является первым **высокоуровневым** объектно-ориентированным интерфейсом, включенным в механизм работы с базами данных Microsoft Jet. Он позволяет VB-программистам работать также с широким кругом

реляционных баз данных через технологию ODBC. DAO наилучшим образом подходит для относительно небольших, локальных баз данных.

Remote Data Objects (RDO).

RDO представляет собой объектно-ориентированный интерфейс доступа к данным, в котором сочетается простой стиль программирования DAO с высокой производительностью и гибкостью низкоуровневых функций ODBC. Доступ к локальным базам данных **Jet** и **ISAM** (Indexed Sequential Access Method) выполняется в RDO через драйверы ODBC, что **снижае**т его быстродействие по сравнению с DAO. Однако RDO обеспечивает возможность работы с большим числом БД различных разработчиков, предоставляя средства доступа к таким элементам, как, например, хранимые процедуры и сложные результирующие наборы данных.

RDO является лишь объектно-ориентированной оболочкой ODBC API, а непосредственный доступ к данным выполняет ODBC драйвер. Объектная модель RDO похожа на DAO, но не требует дополнительной памяти для поддержки локальной базы данных. RDO предоставляет такие дополнительные функции, как серверные **курсоры** (server-side cursors), **отсоединенные** набора записей (disconnected recordsets) и **асинхронную** обработку.

DAO и RDO известны уже достаточно давно, и появление **двух разных** механизмов, как отмечалось выше, было связано с необходимостью оптимизации решения двух отдельных задач: доступа к **локальным** и **удаленным** базам данных соответственно. Однако естественное развитие вычислительных систем привело к необходимости создания единого механизма, который обеспечил бы единый подход при работе с БД различных классов. В результате Microsoft предложила в качестве **единого** интерфейса для доступа к локальным и удаленным данным новую технологию ADO, которая является частью архитектуры Microsoft UDA.

ActiveX Data Objects (ADO).

ADO реализует высокопроизводительный и удобный прикладной интерфейс для OLE DB. Технология ADO нетребовательная к системным ресурсам, создает минимальную нагрузку на сеть и отличается минимальным числом уровней между приложением и источником данных.

ADO предоставляет COM интерфейс Automation, поэтому она поддерживается любой ведущей инструментальной средой быстрой разработки приложений, а также другими инструментальными средствами разработки баз данных, приложений и сценариев.

Разработчики ADO постарались объединить в ней все лучшее, что было в DAO и RDO. Предполагалось, что по мере возможности ADO заменит собой эти технологии.

Часть ADO, носящая название службы удаленных данных (**RDS** - Remote Data Service), отвечает за передачу клиентам отсоединенных наборов записей по протоколу HTTP или Distributed COM (**DCOM**), что позволяет разрабатывать полнофункциональные, ориентированные на работу с данными Web приложения.

ADO работает с источниками данных, имеющих интерфейс OLE DB. С точки зрения программиста, ADO и его расширения являются упрощенным **высокоуровневым** объектно-ориентированным интерфейсом для работы с OLE DB, поддерживая взаимодействие с данными, не имеющими постоянного подключения в сети.

Хотя ADO в целом покрывает функциональность DAO и RDO, она не является совместимой с ними на уровне языка программирования. Это означает, что модернизация уже существующих программ для использования ADO потребует как минимум преобразования кода приложения. В некоторых случая коррекция программ сводится к достаточно простой замене функций, но порой, особенно для наиболее эффективного использования новых возможностей ADO, может потребоваться более серьезная переработка.

Open DataBase Connectivity (ODBC).

ODBC – это открытый интерфейс доступа к базам данных, разработанный фирмой Microsoft. Он представляет собой API довольно **низкого** уровня и предназначен, в основном, для прямого использования в программах, написанных на большинстве современных языков программирования. Несмотря на своё происхождение, этот интерфейс является кроссплатформенным и с успехом работает и в Windows, и в UNIX/Linux, и в MacOS.

ODBC является стандартом Microsoft для работы с **реляционными** данными. Этот компонент архитектуры UDA служит для обеспечения совместимости с более ранними разработками, так как в современных решениях его роль играют собственные провайдеры технологии OLE DB.

Стандартный ODBC интерфейс обеспечивает высокую степень универсальности приложения - один и тот же код применяется для взаимодействия с различными типами СУБД. Это позволяет разработчикам создавать и реализовывать клиент-серверные приложения, не ориентируясь на какую-то определенную СУБД, а значит, не тратить силы и время на учет особенностей конкретных платформ серверов баз данных, с которыми работает приложение. ODBC драйверы – все, что нужно такому приложению для взаимодействия с внешним источником данных. Эти драйверы в соответствии открытым стандартом ODBC создают либо сами поставщики СУБД, либо сторонние разработчики.

Возможности различных СУБД иногда существенно отличаются. Кроме того, в ODBC драйвере допускается реализация только некоторых из всех допустимых функций. По этой причине в ODBC определены три уровня

соответствия драйвера, позволяющие приложению узнавать о наборе функций, доступных в данном конкретном драйвере:

- базовое соответствие (core conformance) – минимум функций, обязательных для всех ODBC драйверов;
- уровень соответствия 1 (level 1 conformance) – включает базовое соответствие и дополнительные функции, которые обычно применяются в СУБД (например, поддержка транзакций);
- уровень соответствия 2 (level 2 conformance) – включает уровень 1 плюс сложные функциональные возможности типа асинхронной работы ODBC.

Механизм работы ODBC поясняется на рис.2.

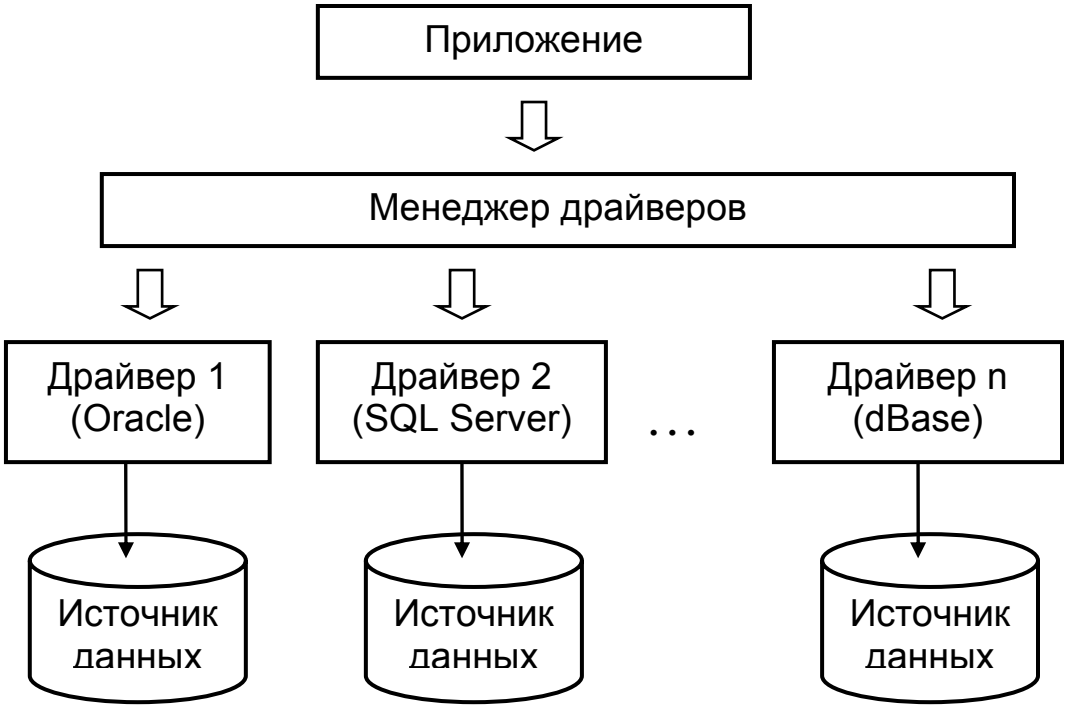

Рис. 2. Схема взаимодействия компонентов ODBC

Не вдаваясь в тонкости реализации ODBC, можно сказать что это, прежде всего, менеджер драйверов ODBC (для платформы Win32 это odbc32.dll). Драйверы непосредственно взаимодействуют с источниками данных. Что это за драйверы и как осуществляется такое взаимодействие - имеет значение только в том случае, если вы собираетесь написать свой собственный драйвер для доступа к источнику данных. Разработчики же СУБД сегодня в большинстве случаев поставляют драйвер ODBC для доступа к своему источнику данных. Таким образом, когда мы хотим получить доступ, например, к MS SQL Server 2000, то нужно установить драйвер MS SQL Server 2000, когда к серверу MySQL – драйвер MySQL и т.д. Стоит отметить, что довольно много драйверов для доступа к самым распространенным источникам данных поставляется вместе с самим ODBC.

Работа с менеджером драйверов ODBC (далее – просто Менеджер) заключается в вызове необходимых функций с определёнными параметрами и в определенной последовательности. В функции Менеджера входит:

- установка и завершение связи с источником данных (сервером БД);
- подготовка и выполнение SQL-операторов;
- получение результатов и навигация по полученным наборам записей, если имеется такая возможность;
- управление транзакциями;
- идентификация ошибок;
- получение различной вспомогательной информации и прочие функции.

Для выполнения этих функций Менеджер вначале должен подготовить некоторые системные ресурсы:

- Идентификатор окружения HENV. Он указывает на область памяти для общей информации (сведения обо всех соединениях с базами данных, информацию о том, какое соединение является текущим и т.п.).
- Идентификатор соединения HDBC. Этот идентификатор указывает на область памяти для информации о конкретном соединении. Идентификатор соединения ассоциируется с единственным идентификатором окружения, в то время как этот идентификатор окружения может иметь несколько связанных с ним идентификаторов соединения.
- Идентификатор оператора HSTMT. Он указывает на область памяти для информации об SQL-операторе. Идентификатор оператора связан с единственным идентификатором соединения. Идентификатор соединения может иметь более одного связанного с ним идентификатора оператора. После закрытия соединения Менеджер должен освободить эти ресурсы и вернуть их системе.

Нет никаких ограничений на использование Access для доступа к функциям этого интерфейса с помощью VBA. Более подробную информацию о способах использования ODBC в Access VBA можно найти по адресу: http://ssy.mccinet.ru/odbc_1.html

Какая бы среда программирования и язык программирования не использовались для организации доступа к реляционным данным посредством ODBC, необходимо произвести настройку ODBC драйверов на использование совместно с требуемыми источниками данных. Для этого служит **утилита ODBC Data Sources** (Источники данных ODBC), доступ к которой можно получить через Панель управления → Администрирование.

С помощью данной утилиты необходимо зарегистрировать новое **имя источника данных** (Data Source Name, DSN). DSN – это уникальный набор сведений, необходимых диспетчеру ODBC драйверов для подключения приложения к конкретной базе данных, поддерживающей ODBC. Имя источника регистрируется в системе, которая будет обращаться к БД, и хранится либо в файле (**файловое DSN**), либо в реестре (**машинное DSN**). Машинное DSN можно задавать как для каждого пользователя

(**пользовательское DSN**), так и для всех пользователей данного компьютера (**системное DSN**).

На рис. 3 приведен пример окна утилиты ODBC Data Source Administrator с несколькими зарегистрированными системными источниками данных.

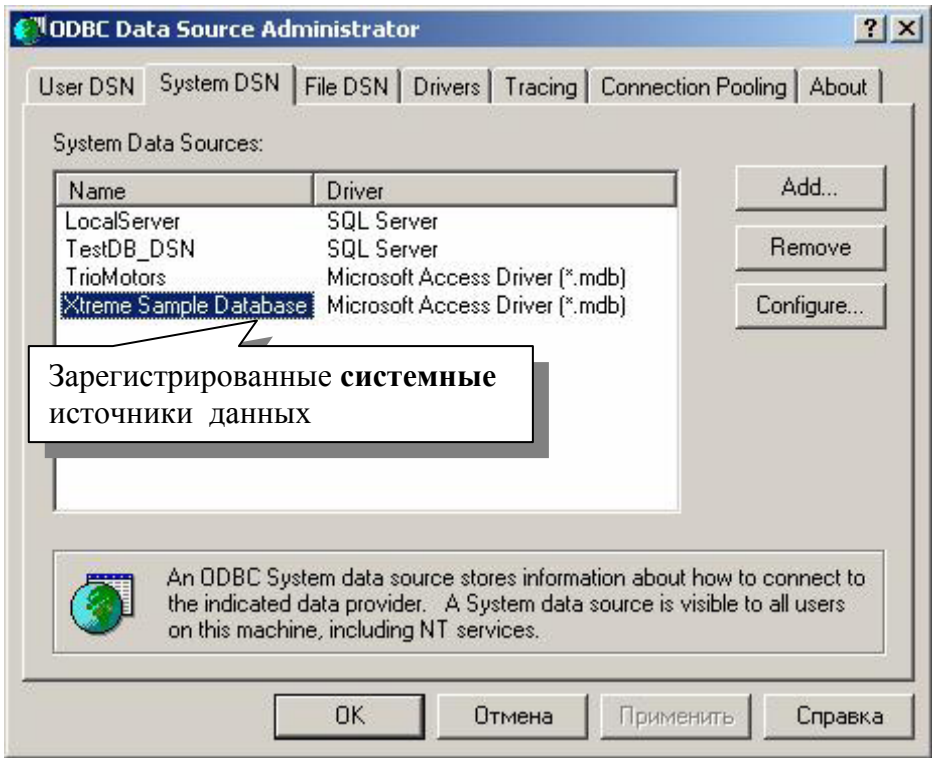

Рис. 3. Источники данных, зарегистрированные с помощью утилиты ODBC Data Source Administrator

В качестве стандарта доступа к данным в ODBC применяется SQL. Когда приложению требуются данных из определенного источника, оно посылает SQL запрос диспетчеру ODBC драйверов, который в ответ загружает соответствующий ODBC драйвер. Тот в свою очередь преобразует поступивший из приложения SQL запрос в понятную СУБД форму SQL и отсылает его серверу БД. Далее СУБД проводит выборку данных и через драйвер и диспетчер возвращает их приложению.

ODBC предоставляет также библиотеку курсоров с курсорами прокрутки для драйверов, поддерживающих базовое соответствие ODBC. Они применяются для просмотра набора записей из базы данных.

ODBC API можно также использовать в языках программирования, например, в C++ для подключения к БД, отправки SQL запросов, получения результатов или ошибок, отключения и т.п. ODBC API – это хорошо документированная технология создания клиент-серверных приложений, но она довольно сложна и требует написания достаточно объемного кода. Поэтому высокоуровневые объектные модели, подобные ADO, применяются значительно чаще.

Object Linking and Embedding DataBase (OLE DB)

OLE DB является **низкоуровневым** интерфейсом для доступа к данным. ADO напрямую взаимодействует с OLE DB. При необходимости приложения могут непосредственно использовать сервисы OLE DB.

OLE DB представляет собой набор COM интерфейсов, предоставляющих приложению единообразный доступ к данным самых различных источников, независимо от их местонахождения или типа. Открытая спецификация OLE DB основана на технологии ODBC, она предоставляет стандарт доступа к данным любого типа. ODBC создавалась для взаимодействия с реляционными БД, а OLE DB разрабатывалась как для реляционных, так и для нереляционных источников, включая БД на мейнфреймах, серверах и персональных компьютерах, а также хранилища файлов и сообщения электронной почты, электронные таблицы, инструментальные средства управления проектами и пользовательские объекты. С помощью предоставляемых OLE DB интерфейсов на основе COM технологии программисты могут создавать дополнительные сервисы баз данных.

Компоненты OLE DB.

В соответствии с принципами построения OLE DB, можно определить три главных компонента: потребители данных (dataconsumers), провайдеры данных (data providers) и провайдеры сервисов (служб) (service providers) (рис. 4).

Любой компонент программного обеспечения, который использует интерфейсы OLE DB, является **потребителем** (consumer) OLE DB. Это может быть бизнес-приложение, инструментальное средство разработки программного обеспечения, например, Borland Delphi или Visual Studio.NET, сложные приложения (sophisticated applications) или же объектная модель ADO, использующая интерфейсы OLE DB. Потребители используют либо те ActiveX Data Objects, которые являются интерфейсом прикладного уровня для обеспечения косвенного (indirect) доступа к данным с применением OLE DB, либо непосредственно OLE DB – для прямого доступа к данным с помощью провайдера OLE DB.

Провайдер (provider) – это часть программного обеспечения, реализующая и предоставляющая набор базовых интерфейсов OLE DB. С точки зрения OLE DB, может быть два вида провайдеров: OLE DB – провайдеры **данных** (data providers) и провайдеры **сервисов** (служб) (service providers).

Провайдер данных (data provider) представляет собой компонент программного обеспечения, «владеющий» данными. Он находится между потребителем и непосредственным массивом данных. В OLE DB все провайдеры представляют данные в табличном формате (в таком же, как реляционные базы данных и электронные таблицы), в виде виртуальных таблиц. Провайдер данных выполняет следующие задачи:

- принимает запросы, поступающие от потребителя, на доступ к данным.
- выполняет выборку или обновление данных из массива данных.
- возвращает эти данные потребителю.

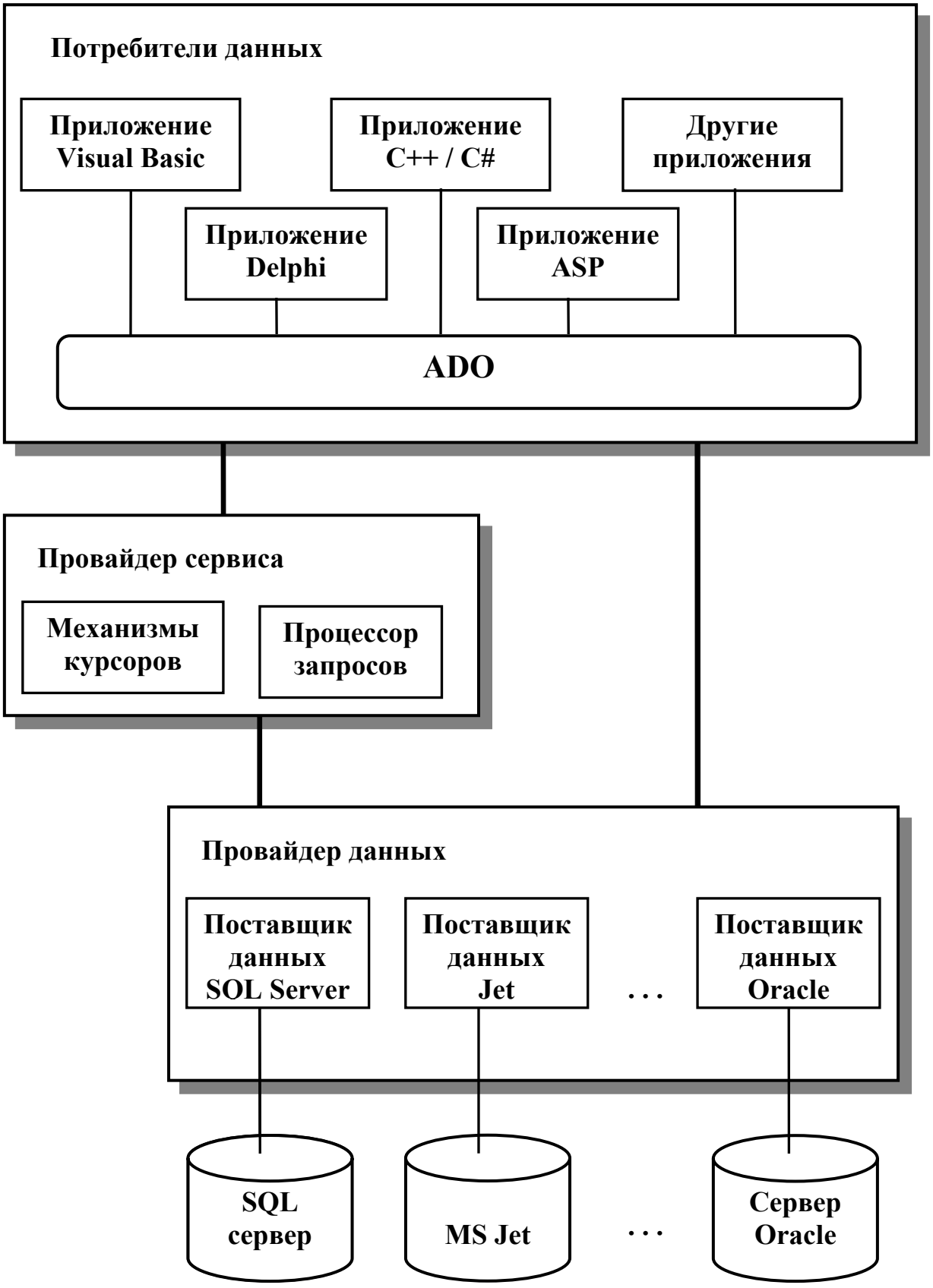

Рис. 4. Компоненты OLE DB

Одним из примеров провайдера данных служит **Microsoft Jet 4.0 OLE DB Provider**. Он используется совместно с механизмом доступа к базам данных Microsoft Jet, применяемым для обработки информации в базах данных Microsoft Access, а также для доступа к информации, упорядоченной с помощью, так называемого инсталлируемого индексно-последовательного метода доступа (Indexed Sequential Access Method, **I-ISAM**), который поддерживается в Jet. К таким данным относятся таблицы, хранимые в рабочих книгах Excel, почтовые файлы Outlook и Microsoft Exchange, таблицы dBase и Paradox, текстовые и HTML-файлы и др. Другим провайдером OLE DB является **Microsoft OLE DB Provider for SQL Server**, используемый для работы с базами данных Microsoft SQL Server.

Провайдер сервисов (service provider) - позже корпорация Microsoft изменила это название на «компонент сервисов» (**service component**) - реализует расширенные функциональные возможности, которые не поддерживаются обычными провайдерами данных, но сам не «владеет» данными. Этот провайдер, например, обеспечивает сортировку, фильтрацию, управление транзакциями, обработку SQL-запросов, функции указателя (курсора).

Провайдер сервисов может напрямую работать с массивами данных или же через соответствующий провайдер данных; в этом случае он выступает в роли потребителя и провайдера. Например, такие провайдеры сервисов, как Microsoft **Cursor Service** for OLE DB и Microsoft **Data Shaping Service** for OLE DB могут интегрироваться с базовыми провайдерами данных OLE DB для расширения их функциональных возможностей.

Отметим, что основная нагрузка в Microsoft UDA ложится на базовые провайдеры данных (native providers) OLE DB, а ODBC используется для обеспечения совместимости с уже существующими приложениями и данными.

Ниже (Табл.1), для справки, приводится список некоторых провайдеров OLE DB.

Таблица 1

Провайдер	**Описание**
Microsoft OLE DB Provider for ODBC Drivers	Позволяет соединяться с любым источником данных ODBC
Microsoft Jet 4.0 OLE DB Provider	Используется для соединения с базами данных Microsoft Access, а также с некоторыми другими источниками данных
Microsoft OLE DB Provider for SQL Server	Применяется для соединения с базами данных Microsoft SQL Server
Microsoft OLE DB Provider for Oracle	Предназначен для получения доступа к базам данных Oracle

Продолжение таблицы 1

Microsoft OLE DB Provider for Internet Publishing	Применяется для соединения с Web серверами и получения доступа к ресурсам, обеспечиваемым Microsoft FrontPage или Microsoft Internet Information Server
OLE DB Provider for Microsoft Directory Services	Используется для соединения с гетерогенными службами каталогов с помощью Microsoft Active Directory Service Interfaces (ADSI)
Microsoft OLE DB Provider for Microsoft Index Server	Обеспечивает программный доступ только для чтения к файловой системе и данным Web, проиндексированным с помощью Microsoft Indexing Service
Microsoft OLE DB Simple Provider	Используется для соединения с простыми текстовыми файлами
Cursor Service for OLE DB	Расширяет функциональные возможности указателя-
Data Shaping Service for OLE DB	Используется для определения отношений между ключами, полями и наборами строк, а также для создания иерархических объектов наборов строк, получаемых от провайдера данных
OLE DB Persistence Provider	Применяется для сохранения данных из наборов строк в формате Advanced Data Table Gram (ADTG) или в формате расширяемого языка разметки — Extensible Markup Language (XML)
Microsoft OLE DB Provider for OLAP Services	Используется вместе с расширениями ADO Multidimensional (ADO MD) для обеспечения доступа к службам OLAP Server for SQL Server 7.0
OLE DB Remoting Provider	Позволяет активизировать провайдеры данных на удаленной машине

Примечание. **Настольные СУБД,** как таковые, не содержат специальных приложений и сервисов, управляющих данными, — взаимодействие с ними осуществляется с помощью файловых сервисов операционной системы. Нередко подобные СУБД имеют в своем составе и средства разработки, ориентированные на работу с данными формата, характерного для этой СУБД, и позволяющие создать более или менее комфортный пользовательский интерфейс (пример тому - офисный Access). Что же касается обработки данных — она целиком и полностью осуществляется в пользовательском (клиентском) приложении.

Развитие настольных СУБД привело к появлению их **сетевых** многопользовательских версий, позволяющих обрабатывать данные, находящиеся в общедоступном хранилище (например, на сетевом диске) нескольким пользователям одновременно. От чисто настольных СУБД их многопользовательские версии отличаются наличием механизма блокировок частей файлов данных (содержащих одну или несколько записей таблицы), что позволяет обращаться к одному и тому же файлу нескольким пользователям одновременно.

Недостатки подобных СУБД **не очевидны** и становятся заметны, как правило, при росте хранимых объемов данных и увеличении числа пользователей. Обычно они проявляются в снижении производительности и в возникновении сбоев при обработке данных после некоторого времени использования клиентских приложений. Причина подобных проблем кроется в основном принципе работы таких СУБД и основанных на них информационных систем, заключающемся в обработке данных внутри пользовательского приложения. Например, если с помощью такой системы требуется выполнить запрос согласно какому-либо критерию (например, выбрать заказы, обработанные за последние два часа, из таблицы заказов), то, в лучшем случае (если эта таблица проиндексирована по времени поступления заказа), приложение должно прочесть с сетевого диска весь индекс, найти в нем сведения о местоположении записей в файлах, содержащих таблицу, и затем прочесть эти части файлов. В общем же случае, когда таблица не проиндексирована по данному полю, ее необходимо загрузить с сетевого диска и анализировать.

Еще одна проблема настольных СУБД заключается в возможности нарушения ссылочной целостности данных, так как единственным механизмом, контролирующим ее, является пользовательское приложение. Поэтому все пользовательские приложения должны содержать соответствующий код, и доступ к файлам базы данных из любых других приложений должен быть запрещен. В наиболее популярных настольных СУБД (например, Microsoft Access, Corel Paradox) код, контролирующий стандартную ссылочную целостность, содержится в библиотеках, используемых всеми приложениями, работающими с этой базой данных, а сама база данных при этом может содержать описание правил ссылочной целостности.

Следующим этапом развития СУБД для персональных компьютеров были так называемые **серверные** СУБД. Кратко объясним, чем серверные СУБД отличаются от настольных.

Архитектура «клиент/сервер», для которой предназначены серверные СУБД, является в определенной степени возвратом к прежней «мэйнфреймовой» модели, основанной на централизации хранения и обработки данных на одном выделенном компьютере, где функционирует специальное приложение или сервис, называемый **сервером баз данных**. Сервер баз данных отвечает за работу с файлами базы данных, поддержку ссылочной целостности, резервное копирование, обеспечение авторизованного доступа к данным, протоколирование операций и, конечно, за выполнение пользовательских запросов на выбор и модификацию данных и метаданных. Клиентские приложения, являющиеся источниками этих запросов, функционируют на персональных компьютерах в сети.

Не останавливаясь подробно на достоинствах и недостатках подобной архитектуры, отметим лишь, что при использовании серверных СУБД выполнение запросов производится самим сервером, поэтому клиентские приложения получают от сервера только результаты самого запроса и не требуют передачи всего индекса или всей таблицы, что существенно снижает сетевой трафик при обработке запросов. Отметим также, что многие объекты, предназначенные для реализации бизнес-правил, такие как хранимые процедуры и триггеры, доступны лишь в серверных СУБД.

ActiveX Data Objects. NET (ADO. NET)

В настоящее время с появлением и активным развитием платформы Framework дальнейшее развитие получила и технология ADO, которая стала более универсальной и ориентированной на создание распределенных приложений. Универсальность ее достигается во многом благодаря использованию языка XML, являющегося сегодня стандартом обмена данными между различными системами и средами.

Рассмотрим коротко архитектуру и основные принципы взаимодействия служб ADO.NET. Основные компоненты ADO.NET и их взаимодействие в процессе обработки данных (по MSDN) представлены на рис.5.

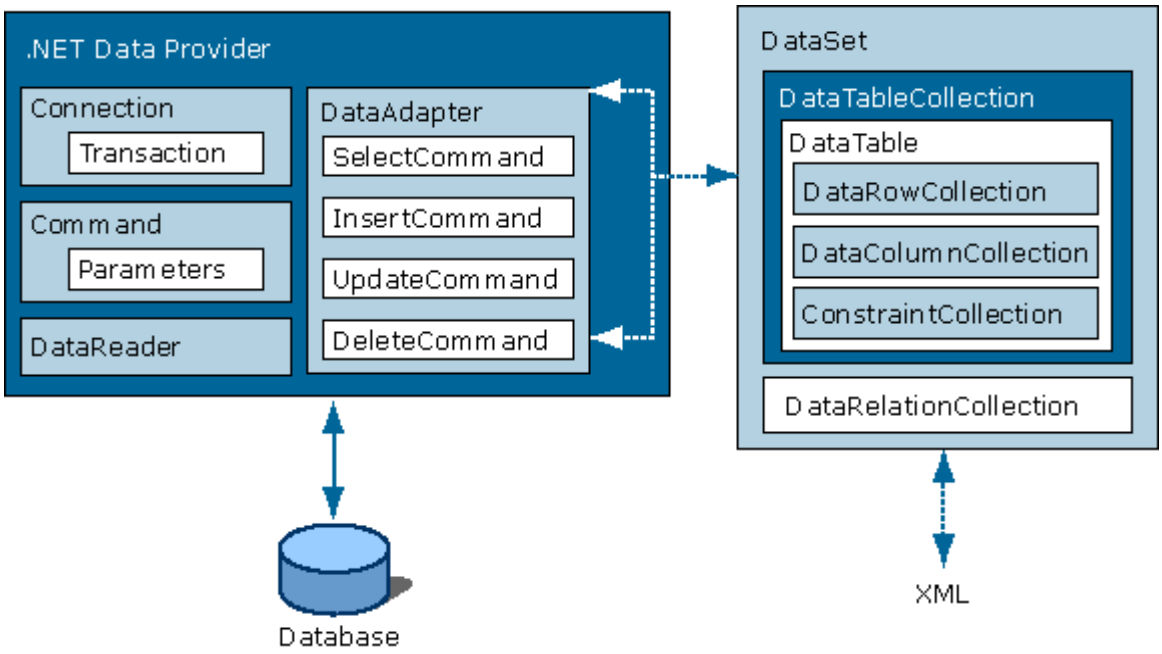

Рис.5. Основные компоненты архитектуры ADO.NET

Как видно из рисунка, доступ к данным основан на использовании двух компонентов: **набора данных (**DataSet**)**, в котором данные хранятся на локальном компьютере и **провайдера данных** (Data Provider), выполняющего функции посредника при взаимодействии программы с БД.

Объект DataSet – это представление в памяти компьютера данных, изолированных от источника данных. Этот объект можно также рассматривать как локальную копию фрагмента БД. В DataSet данные можно загрузить из любого допустимого источника, например из БД SQL Server, Microsoft Access или XML-файла. Объект DataSet хранится в памяти, его содержимым разрешено манипулировать и обновлять независимо от БД, играющей роль источника данных. При необходимости, объект DataSet может служить шаблоном для обновления серверной БД.

Объект DataSet (см. рис.6) содержит набор объектов DataTable (этот набор может быть и пустым, то есть не содержать ни одного объекта DataTable). Каждый объект DataTable представляет в памяти компьютера одну таблицу.

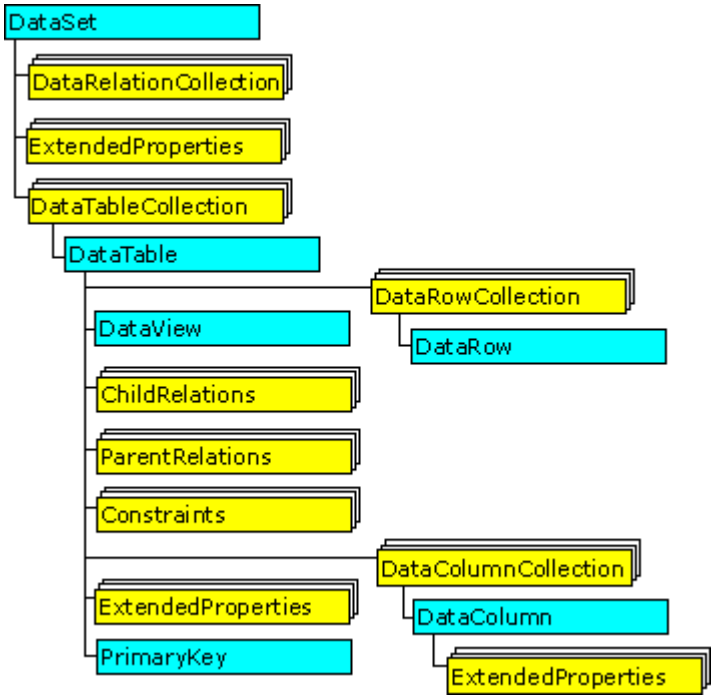

Рис.6. Внутреннее устройство объекта DataSet.

Структура объекта DataTable определяется двумя наборами: DataColumns, куда входят все столбцы таблицы, которую представляет объект DataTable, и набором ограничений (Constrains) таблицы. Вместе эти два набора составляют схему таблицы.

В DataTable также входит набор Data Rows, где, собственно, и хранятся данные объекта DataSet. Конечно, структура объекта DataSet сложнее, чем та, которая приводится здесь и состоит из гораздо большего числа элементов, однако целью данного рассмотрения не является детальное рассмотрение объектной структуры ADO.NET, а лишь знакомство с ее основными элементами.

Связь с БД создается и поддерживается при помощи провайдера данных. В действительности провайдер – это набор взаимосвязанных компонентов, обеспечивающих эффективный высокопроизводительный доступ к данным. В составе .NET Framework поставляются **два типа** провайдеров данных: SQL Server .NET Data Provider (SQL **Managed** Provider), созданный для работы с SQL Server версии 7.0 и выше, и OleDb .NET Data Provider (ADO **Managed** Provider) – для подключения к OLE DB-источникам данных.

SQL Server .NET Data Provider оптимизирован для работы с Microsoft SQL Server. Он работает по специальному протоколу, называемому Tabular Data Stream (TDS) и не использует ни ADO, ни ODBC, ни какую-либо другую технологию. Ориентированный специально на MS SQL Server, этот протокол позволяет увеличить скорость передачи данных и тем самым повысить общую производительность приложения. За это приходится платить потерей переносимости на другие сервера баз данных, вследствие использования в TDS специализированных классов.

Находится этот провайдер в пространстве имен System.Data.SqlClient.

OLE DB .NET Data Provider это более общий провайдер, способный работать с почти любым источником данных, поддерживающим OLE DB. (исключения представлены ниже). Поскольку для своей работы он должен использовать COM, то между провайдером и источником данных в этом случае присутствуют две прослойки: обертка, позволяющая managed-приложению использовать COM, и собственно провайдер OLE DB для COM. Естественно, в этом случае производительность уступает предыдущему решению, но сохраняется возможность использования практически любых источников данных.

Находится этот провайдер в пространстве имен System.Data.OleDb.

К числу провайдеров, прошедших тестирование Microsoft на возможность работы в ADO .NET, относятся следующие OLE DB-провайдеры:

Driver	Provider
SQLOLEDB	Microsoft OLE DB Provider for SQL Server
MSDAORA	Microsoft OLE DB Provider for Oracle
Microsoft.Jet.OLEDB.4.0	OLE DB Provider for Microsoft Jet

Не поддерживается OLE DB Provider for ODBC (MSDASQL), а также OLE DB version 2.5 (в частности, Microsoft OLE DB Provider for Exchange и Microsoft OLE DB Provider for Internet Publishing). Для корректной работы провайдера OLE DB следует установить MDAC 2.6 и выше.

Не рекомендуется использование баз данных Microsoft Access для многоуровневых приложений.

Для работы **с источниками данных ODBC** в среде ADO .NET, Microsoft предоставляет соответствующую поддержку - провайдер ODBC .NET Data Provider. Этот **третий** тип поставщика данных .NET (ODBC **Managed** Provider) можно загрузить с сайта http://www.microsoft.com/downloads/

На Рис. 7 показаны различные конфигурации подключений, по которым приложение может связываться с базой данных через ADO.NET. При выборе пути сначала определяется, какой поставщик данных .NET будет использоваться. Если это SQL Server 7.0 или более поздняя версия, то подключается поставщик данных SQL Server.NET. Если база данных SQL Server 6.5 или отличная от SQL Server (например, Oracle), понадобится поставщик данных OLE DB .NET. Заметим, что можно задействовать поставщик данных OLE DB .NET для баз данных SQL 7.0 и выше, но тогда потеряется выигрыш в производительности, который дает прямое подключение к SQL Server через протокол TDS. Однако в этом неспецифическом способе есть свой плюс — мобильность, т. е. можно менять базы данных без модификации кода.

Далее необходимо определить, какую задачу требуется выполнить. Если надо просто прочитать и отобразить данные из источника данных, - объекта

DataReader вполне достаточно. Но если предстоит манипулировать данными (например, редактировать или удалять), нужно использовать объект DataSet. Задействовать этот объект целесообразно только в случае необходимости, потому что он работает медленнее, чем DataReader (DataSet использует DataReader для заполнения таблиц).

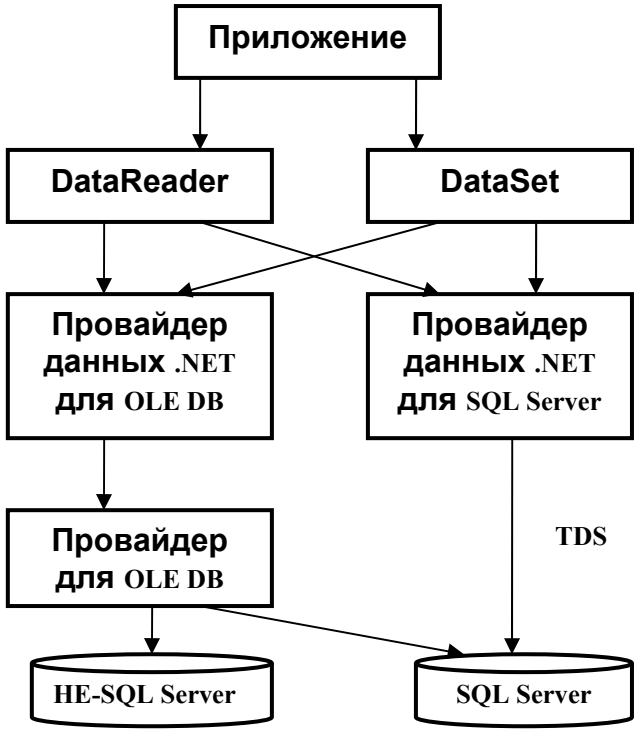

Рис.7. Конфигурации подключений к источникам данных в ADO.NET

Любой провайдер данных состоит из близких версий следующих универсальных классов компонентов:

- Connection – обеспечивает подключение к БД;
- Command – применяется для управления источником данных; позволяет исполнять команды, не возвращающие данные, например, INSERT, UPDATE и DELETE, либо команды, возвращающие объект DataReader (такие, как SELECT);
- DataReader – предоставляет доступный только для однонаправленного чтения набор записей, подключенный к источнику данных;
- DataAdapter – заполняет отсоединенный объект DataSet или DataTable и обновляет его содержимое.

Доступ к данным в ADO.NET осуществляется так: объект Connection устанавливает между приложением и БД соединение, напрямую доступное объектам Command и DataAdapter. Объект Command позволяет исполнять команды непосредственно над БД. Если исполненная команда возвращает несколько значений, Command открывает к ним доступ через объект DataReader. Полученные результаты можно обрабатывать напрямую, используя код приложения, либо через объект DataSet, заполнив его при помощи объекта DataAdapter. Для обновления БД также применяют объекты

Command или DataAdapter.

Вот текст программы на языке C#. Для простоты она оформлена в виде консольного приложения.

```
using System;
using System.Data;
using System.Data.OleDb;
namespace ConsoleApplication1
{
    class CAdoNetSample
    {
        [STAThread]
        static void Main(string[] args)
        {
                // строка соединения; настройте ее на расположение базы
                //данных Борей.mdb в вашей системе
            string strConn=
            "Provider=Microsoft.Jet.OLEDB.4.0;Password="";User ID=Admin;"+
            "Data Source=C:\Program Files\Microsoft Office\Office10\Samples\Борей.mdb;";
                // создаем Connection и открываем его
            OleDbConnection cnn = new OleDbConnection(strConn);
            cnn.Open();

                // создим команду SQL, которая выберет все страны, в которых
                // живут клиенты, и упорядочим их список по алфавиту. Обратите
                // внимание - создаваемая команда связана с соединением cnn
            OleDbCommand cmd = cnn.CreateCommand();
            cmd.CommandText=
            "SELECT DISTINCT Страна FROM Клиенты ORDER BY Страна";

                // поскольку мы намереваемся читать список последовательно,
                // то нас вполне устроит функциональность объекта DataReader.
                //Выполняем команду, результаты ее выполнения будут доступны
                // через rdr
            OleDbDataReader rdr = cmd.ExecuteReader();

                // выдаем на печать все полученные результаты
            while (rdr.Read())
            {
                Console.WriteLine(rdr.GetString(0));
            }
                // освобождаем занятые ресурсы
            rdr.Close();
            cnn.Close();
        }
    }
}
```

Можно создать в среде Visual Studio .NET пустой проект консольного приложения, скопировать в него текст программы и убедиться в её работоспособности.

Объекты прямого доступа к данным DAO

Как модель программирования, основанная на объектах, DAO предоставляет для управления отдельными компонентами БД различные коллекции (семейства) объектов, свойства и методы. Объекты доступа к данным позволяют создавать программы, обеспечивающие прямой доступ и обработку данных в локальных и удаленных БД, а также изменять структуру БД.

Для использования модели DAO в Access необходимо подключить соответствующие библиотеки. Для этого в главном меню инструментальной среды VBA необходимо выбрать команду **Tools\References** и в открывшемся окне установить флажок напротив строки Microsoft DAO 3.6 Object Library, затем нажать кнопку OK (рис.7). Если в проекте Access необходимо использовать несколько технологий доступа к данным совместно (например, DAO и ADO), необходимо выбрать всех их. Используемая по умолчанию технология в этом случае будет определяться приоритетом (порядком следования ее в списке). При необходимости использования другой технологии потребуется прямая ссылка на нее в тексте программы.

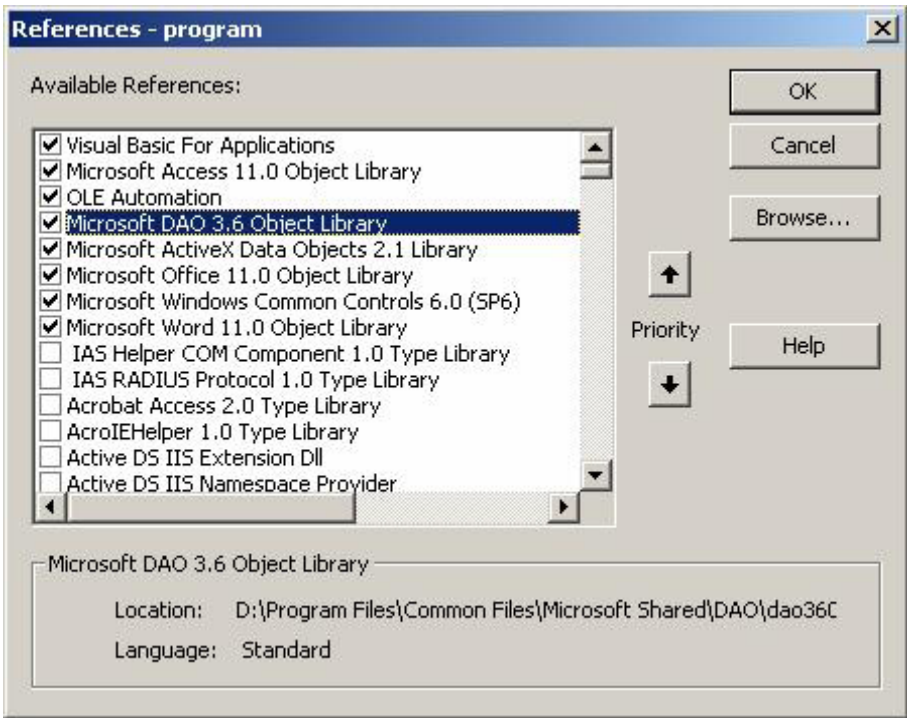

Рис. 7. Окно подключения библиотек расширения в Access.

Microsoft Access может быть использован, с одной стороны, в качестве **настольной СУБД** и составной части пакета MS Office, а с другой стороны, в качестве **клиента Microsoft SQL Server**, позволяющего осуществлять его администрирование, манипуляцию его данными и создание приложений для этого сервера.

Для реализации этих функций объектная модель DAO поддерживает **две среды** работы с базами данных (называемых, также рабочими областями или рабочими пространствами).

- **Рабочая область MS Jet** обеспечивает доступ к данным в базах данных MS Jet, в базах данных ODBC, подключенных к ядру MS Jet, а также к устанавливаемым (инсталлируемым) источникам данных ISAM других форматов.

Рабочую область MS Jet используют при открытии базы данных Microsoft Jet (файл *.mdb) или прочих баз данных, выполняющихся на **настольных** системах, а также в ситуациях, когда требуются уникальные характеристики Microsoft Jet, такие как возможность объединения данных из баз данных различных форматов.

- **Рабочая область ODBCDirec**t позволяет получить доступ к СУБД через ODBC без загрузки и использования ядра Microsoft Jet. ODBCDirect предоставляет возможность выполнять запросы или хранимые процедуры на **сервере**, а так же использовать специфические возможности ODBC, такие как пакетное обновление и асинхронное исполнение запросов.

Структура объектной модели DAO **зависит** от используемого рабочего пространства. Различия касаются в основном возможности управлять пользователями и метаинформацией (структурой БД) в Jet. Иерархия классов объектной модели DAO для рабочих областей MS Jet и ODBCDirect изображена на рис. 8 и рис.9, соответственно. Рассмотрим наиболее часто используемые объекты.

Объектом самого верхнего уровня в DAO (корневым объектом) является **DBEngine**. В нем содержится две коллекции: **Errors** (коллекция ошибок) и **Workspaces** (рабочие пространства).

Любые операции, связанные с объектами DAO, могут вызвать одну или несколько ошибок. В случае возникновения ошибок они помещаются в коллекцию **Errors**. Ошибки поступают по мере возникновения, т.е. самая последняя ошибка является последним элементом коллекции; она же соответствует объекту Err в VBA. Коллекция Errors не содержит методов Append и Delete, поэтому ошибки не могут быть добавлены в коллекцию иным способом. Каждый элемент Error коллекции Errors описывает одну ошибку.

Коллекция **Workspaces** содержит активные объекты Workspace. Объект Workspace определяет именованную сессию (сеанс) пользователя. Он содержит открытые базы данных, предоставляет механизмы для проведения транзакций, а также методы для управления базами данных, пользователями группами и подключениями.

При открытии базы данных Access, объект DBEngine сначала создает коллекцию (семейство) Workspaces и объект Workspace по умолчанию. При этом тип создаваемой рабочей области определяется значением свойства DefaultType объекта DBEngine.

Если при открытии базы данных **используются** средства защиты (подключение к базе данных производится с помощью специального окна диалога Подключение), то Access запросит ввод пароля и идентификатор

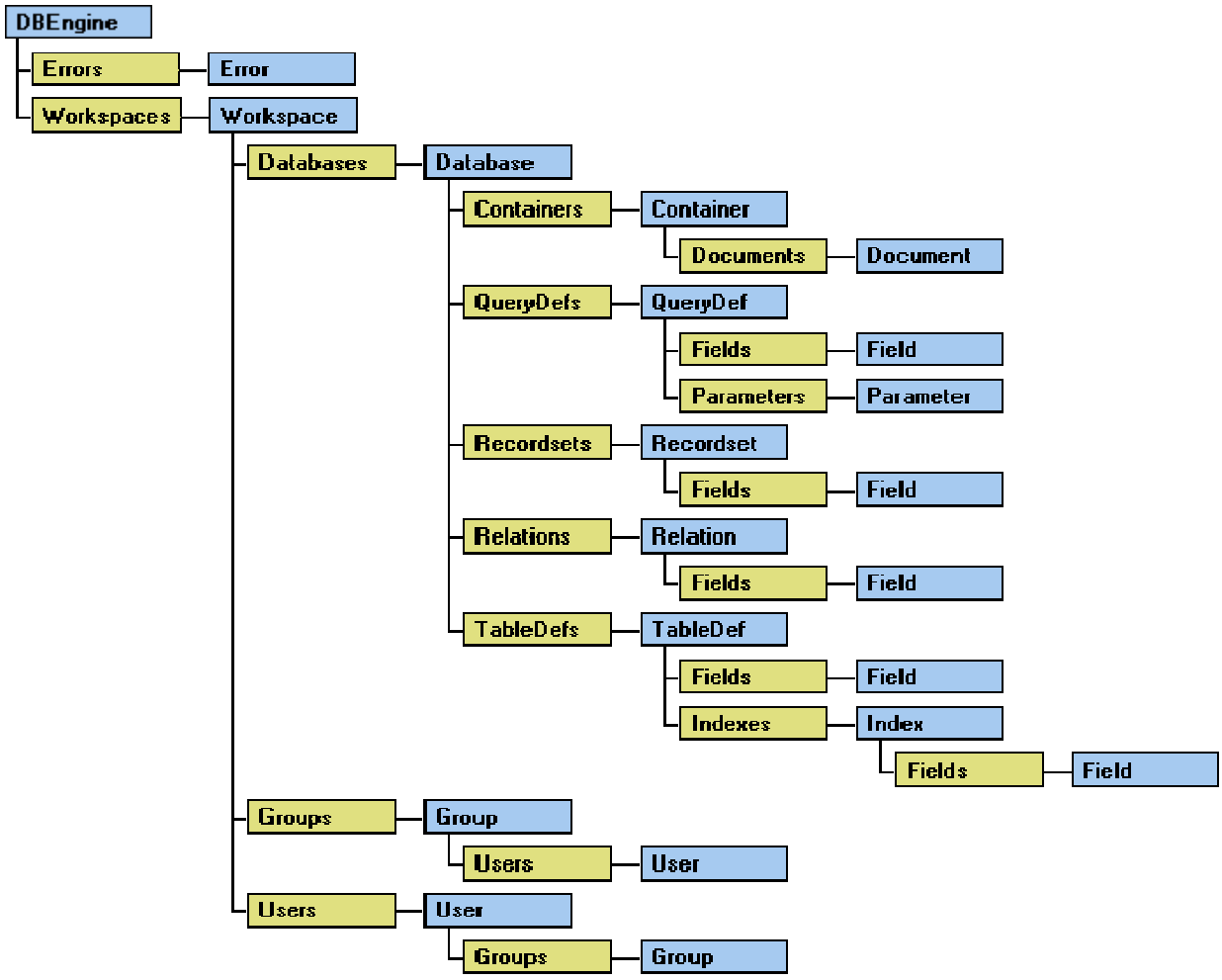

Рис.8. Объектная модель DAO для рабочей области Microsoft Jet.

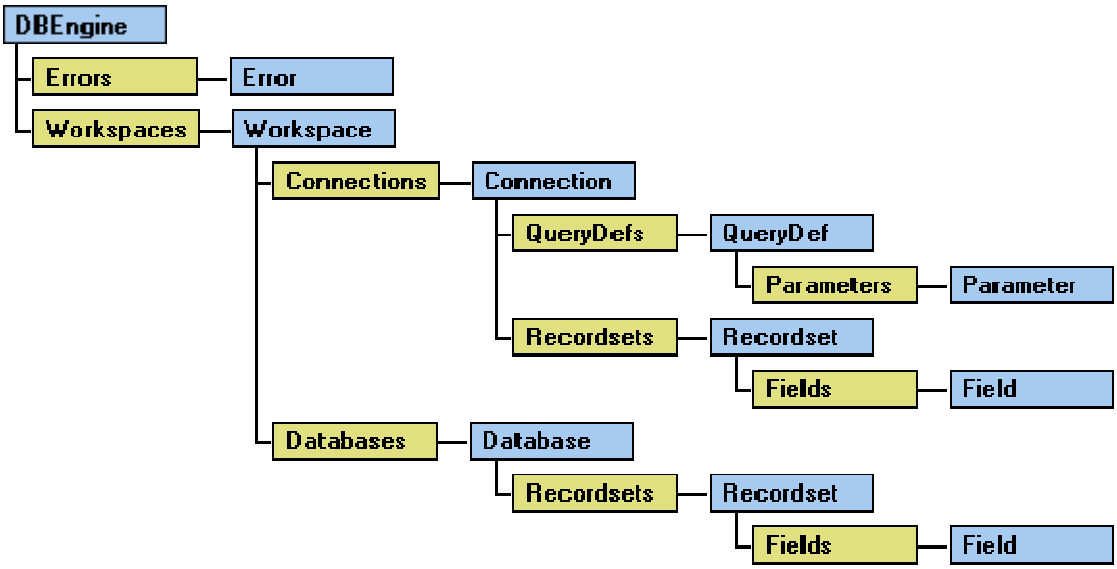

Рис. 9. Объектная модель DAO для рабочей области ODBCDirect.

пользователя. В этом случае, в объекте Workspace, установленном по умолчанию, DBEngine сможет создать объекты **User** (Пользователь) и **Group** (Группа).
Если средства защиты **не используются**, DBEngine создает по умолчанию объект User с именем Admin и объект Group с именем Admins.
Далее DBEngine создает в объекте Workspace, установленном по умолчанию, объект Database (База данных) в составе семейства **Databases**. DBEngine использует информацию о текущем объекте User и/или Group для того, чтобы определить, имеет ли пользователь право доступа к объектам базы данных.

С помощью процедур VBA можно создать дополнительные объекты в коллекции **Databases**, открывая файлы с расширением ***.**mdb. Каждый открытый объект Database содержит коллекцию **Container**s (Контейнеры), которая совместно с семейством **Documents** используется DBEngine для хранения определений всех таблиц, запросов, форм, отчетов, макросов и модулей.

Коллекция **TableDefs** (Таблицы) используется для просмотра и изменения существующих таблиц. Можно создать новые таблицы внутри этого семейства.

Аналогично, коллекция **Relation**s (Отношения) содержит объекты Relation (Отношение), определяющее связи между таблицами и режимы обеспечения целостности данных, применяемые к таблицам.

Коллекция **QueryDefs** (Запросы) включает в себя объекты QueryDef (Запрос), содержащие определения запросов к базе данных. Можно изменять существующие запросы и создавать новые.

Коллекция **Connections** (Соединения) на рис.9 используется для подключения к БД с использованием драйверов ODBC. Фактически, объект Connection и объект Database представляют две разные ссылки на один и тот же объект, так как свойства каждого из этих объектов позволяют получить ссылку на второй объект. **Свойство Database** объекта Connection и **свойство Connection** объекта Database облегчают модификацию подключения к ODBC источнику данных приложения-клиента, использующего ядро MS Jet, для использования технологии ODBCDirect.

Наконец, коллекция **Recordsets** (Наборы записей) содержит объекты Recordset (Набор записей) для каждого открытого в базе данных набора записей (**более подробно об объектах Recordset см. ниже - "Дополнительные сведения"**).

При программировании работа с объектами происходит с помощью **объектных переменных**, существенно упрощающих конструкцию ссылок. При этом Access гораздо быстрее обрабатывает ссылки, использующие объектную переменную, чем полностью заданные имена.
Для объявления объектной переменной в VB (VBA) используется инструкция Set, имеющая синтаксис:

Set *имя_переменной = ссылка_на_объект*

Примечание. Предварительно объявите *имя_переменной* в инструкциях Dim, Public или Static как объект одного из **типов:** Application, Container, Control, Database, Document, Error, Field, Form, Group, Index, Parameter, Property, QueryDef, Recordset, Relation, Report, TableDef, User, Workspace. Тип объектной переменной должен быть совместим с типом объекта, на который указывает *ссылка_на_объект.*

Для описания объекта на нижнем уровне в *ссылке_на_объект* можно использовать другую объектную переменную (см. ниже пример). Кроме того, можно использовать метод некоторого объекта для создания нового объекта в семействе и назначить этот объект объектной переменной.

Следует иметь в виду, что объектная переменная содержит ссылку на некоторый объект, а не на его копию. Если несколько объектных переменных указывают на один и тот же объект, то изменение любого свойства этого объекта сразу отразится на всех переменных, ссылающихся на него. Единственное исключение из этого правила: несколько переменных типа Recordset могут ссылаться на один и тот же набор записей, но при этом их свойства Bookmark (Закладка) могут указывать на различные записи этого набора.

Приведем ряд **примеров**, иллюстрирующих работу с объектами DAO.

● Создадим с помощью объекта Workspace рабочую область MS Jet, либо рабочую область ODBCDirect. Для выполнения данной операции сначала опишем переменную типа объекта Workspace, например, wrkSpace, а затем свяжем эту переменную с текущим сеансом:

```
Dim wrkSpace as Workspace
Set wrkSpace = CreateWorkspace (name, user, password, type)
```

Параметр (аргумент) *name* - строка, содержащая уникальное имя нового объекта Workspace. В данном случае для доступа к рабочей области используется объектная переменная, поэтому данный параметр можно опустить.

Параметр *user* - строка, задающая владельца создаваемого объекта Workspace.

Параметр *password* - строка, содержащая пароль для нового объекта Workspace.

Параметр *type* - константа, задающая тип рабочей области. Значениями данного параметра являются константы:

■ Константа dbUseJet создает рабочую область MS Jet, используемую для прямого доступа к источнику данных с помощью ядра базы данных Jet.

■ Константа dbUseODBC создает рабочую область ODBCDirect. При этом осуществляется прямой доступ к источнику данных ODBC без использования ядра базы данных Jet.

Если опустить необязательный параметр *type*, то тип создаваемой рабочей области определяется по умолчанию, как уже отмечалось выше.

Пусть разрабатываемое приложение должно извлечь данные, находящиеся в файле Access. Тогда для доступа к базе данных используем следующий фрагмент:

```
Dim wrkJet as Workspace
Set wrkJet = CreateWorkspace (name:=" ", user:="admin", _
            password:=" ", type:=dbUseJet)
```

● В созданной рабочей области открывается требуемая база данных с помощью метода *OpenDatabase* объекта Workspase. При открытии база автоматически добавляется в семейство Databases.

```
Dim db_var_name as Database
Set db_var_name = wrkSpace.OpenDatabase (dbname, [options], [read-only], [connect])
```

Параметр *dbname* - строка, задающая полное имя существующего файла базы данных.

Необязательный аргумент *options* позволяет задать параметры режима работы с открываемой базой данных. Допустимые значения для данного аргумента определяются типом используемой рабочей области.

Для рабочей области MS Jet:

■ Константа *True* определяет открытие базы данных в монопольном режиме, при котором запрещается изменение информации в базе другим пользователям.

■ Константа *False* (по умолчанию) позволяет открыть базу данных для общего доступа. При этом допускается одновременное внесение изменений несколькими пользователями.

Для рабочей области ODBCDirect параметр *options* задает вывод приглашения пользователю на установку подключения. Допускается использование одной из следующих констант:

■ Константа *dbDriverNoPrompt* определяет использование Диспетчером драйвера ODBC строки подключения, которая задается аргументами *dbname* и *connect* .

Примечание. Константа *dbDriverNoPrompt* указывается только в том случае, если строка подключения полностью определяется параметрами *dbname* и *connect* .В противном случае возникает ошибка.

■ Константа *dbDriverPrompt* определяет открытие Диспетчером драйвера ODBC диалогового окна ODBC Data Sources (Источники данных ODBC), в котором отображаются данные, определяемые аргументами *dbname* и *connect.*

■ Константа *dbDriverComplete* (по умолчанию для рабочей области ODBCDirect) используется, если в аргументах *dbname* и *connect* указаны все, необходимые для подключения к источнику данных ODBC, сведения.

■ Константа *dbDriverCompleteRequired* определяет действия, аналогичные задаваемым константой *dbDriverComplete* с тем отличием, что драйвер ODBC отключает отображение приглашений на ввод сведений, не являющихся необходимыми для подключения.

Необязательный параметр *read-only* имеет значение *True*, если база данных открывается только для чтения, и значение *False* (по умолчанию) при открытии базы данных с доступом для чтения и записи.

Параметр *connect* содержит сведения о подключении, в том числе и пароли.
Следующий фрагмент кода иллюстрирует использование метода *OpenDatabase* в частном случае:

```
Dim db as Database
Set db = wrkSpace.OpenDatabase ("C:\Program Files\MicrosoftOffice\Office\Samples_
        Борей.mdb, options:=False, read_only:=False)
```

● Объектная переменная ссылается на текущую базу данных:

```
Dim dbMyDB as Database
Set dbMyDB = CurrentDb ( )
```

● Ссылка на таблицу *tblClubs* в текущей базе данных, используя описанную переменную *dbMyDB* в предыдущем примере:

```
Dim tblMyTable as TableDef
Set tblMyTable  = dbMyDB![tblClubs]
```

Примечание. Обратите внимание на то, что в приведенном выше примере, нет необходимости явно ссылаться на семейство TableDefs базы данных (полная ссылка - dbMyDB.TableDefs![tblClubs]), так как TableDefs - это назначаемое по умолчанию семейство в объекте Database. Access предполагает, что [tblClubs] ссылается на имя объекта в назначаемом по умолчанию семействе базы данных.
Таким образом, при ссылке на объект, содержащийся в назначаемом по умолчанию или единственном семействе, **не обязательно** указывать имя семейства.

● Ссылка на поле *Notes* в таблице *tblClubs*, используя описанную переменную *tblMyTable* в предыдущем примере:

```
Dim fldMyField as Field
Set fldMyField = tblMyTable! [Notes]
```

● Обычно для работы с БД достаточно определить объектную переменную, типа *db As DAO.Database*. Но если необходимо использовать **транзакции**, то в этом случае не обойтись без объявления Workspace (рабочего пространства). Вот небольшой пример декларации и подключения к БД:

```
Dim objWorkspace As DAO.Workspace
Dim objDatabase As DAO.Database
Dim objRecordset As DAO.Recordset
...
Set objWorkspace = DBEngine.Workspaces(0)
Set objDatabase = objWorkspace.OpenDatabase(...)
Set objRecordset = objDatabase.OpenRecordset(...)
...
objWorkspace.BeginTrans      'Начало транзакции
With objRecordset
  Do Until .EOF
    .Edit
```

```
    .Fields("field1") = "test"
    .Update
    .MoveNext
  Loop
End With
objWorkspace.CommitTrans   'Окончание транзакции
```

Не рекомендуется использовать транзакции без необходимости, так как они требуют значительных ресурсов. Кроме того, в многопользовательской системе необходимо, в этом случае, учитывать влияние транзакций, выполняемых другими пользователями.

Все объекты Database, открытые в объекте Workspase, имеют общую область определения транзакций. Это означает, что вызов методов BeginTrans, CommitTrans или Rollback для объекта Workspace затрагивает все открытые базы данных в этом объекте Workspace. Поэтому, если необходимо независимое использование транзакций, следует открыть дополнительно объект Workspace, а в нём открыть экземпляр-копию объекта Database.

Примечание. Вы можете создать несколько копий базы данных в коллекции Databases. При этом каждой копии объекта Database следует назначить соответствующую объектную переменную для ссылки.

Технология DAO предусматривает **совместное** использование рабочих областей MS Jet и ODBCDirect в приложении, обеспечивающих различные, но дополняющие друг друга функциональные возможности. При этом следует ориентироваться на их относительное (друг перед другом) преимущество в предоставлении требуемой приложению функциональности.
ODBCDirect предоставляет следующие преимущества при выполнении операций доступа к источникам данных ODBC:

- Увеличение производительности за счет отсутствия необходимости в загрузке ядра БД MS Jet, а также выполнения обработки запросов сервером ODBC, а не клиентским приложением.
- ODBCDirect поддерживает асинхронные запросы, исключающие необходимость ожидания завершения запроса, чтобы выполнить следующую операцию.
- Поддержка пакетного изменения, позволяющего кэширование изменения объекта Recordset на месте, а затем передачу серверу всех изменений в едином пакете.

Вместе с тем, не все операции языка определения данных (DDL) реализуются в ODBCDirect (например, не поддерживается объект TableDef). Кроме того, данные, к которым осуществляется доступ с помощью рабочей области ODBCDirect, не могут быть привязаны к формам или элементам управления приложения. Также отсутствует поддержка обновляемых объединений и неоднородных объединений нескольких таблиц в объектах Recordset. Тем не менее, если перечисленные возможности не требуются, то лучше использовать рабочую область ODBCDirect.

Объекты прямого доступа к данным ADO

ADO является преемником DAO. Однако в этой технологии реализована другая объектная модель: объектов стало меньше, но при этом увеличилось число свойств, методов, аргументов и событий.

Объектная модель ADO призвана обеспечить доступ к наиболее часто применяемым функциям OLE DB. ADO состоит из трех **основных** компонент: объекта **Connection**, объекта **Command** и объекта **Recordset** (рис. 10).

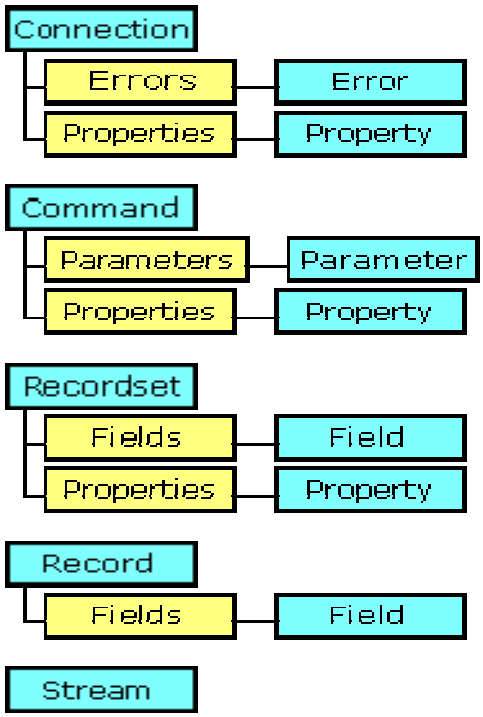

Рис. 10. Объектная модель ADO (версия 2.6)

Отличие объектной модели ADO от DAO состоит в том, что многие ее объекты **независимы** друг от друга. Иерархия объектов ADO допускает создание только непосредственно нужных объектов, когда экземпляры Recordset, Connection и Command порождаются напрямую без создания их родителей. Например, можно создать Recordset без явной инициализации объекта Connection – ADO сделает это самостоятельно.

Объект **Connection** устанавливает соединение между приложением и внешним источником данных, например SQL Server. Объект задает сеанс работы с источником данных. В случае, когда речь идет об удаленном источнике и клиент-серверном приложении, установление соединения означает физическое подключение к серверу в сети.

Большинство свойств и методов этого объекта определено для всех стандартных провайдеров. Тем не менее, есть специфика, определяемая каждым конкретным провайдером.

Используя методы и свойства этого объекта, можно выполнить следующее:

- Конфигурировать подключение перед его фактическим открытием с помощью свойств ConnectionString, ConnectionTimeout, Mode. Первое из этих свойств имеет статус свойства по умолчанию.
- Обращаться на клиентской стороне к службе управления курсором - Cursor Service for OLE DB, задавая adUseClient в качестве значения свойства CursorLocation.
- Указать базу данных, используемую по умолчанию, с помощью свойства DefaultDatabase; аналогично, свойство Provider устанавливает провайдера по умолчанию.
- Используя методы Open и Close, многократно открывать и закрывать соединение с возможностью изменения его параметров от сеанса к сеансу.
- Используя метод Execute, выполнить команду, не обращаясь к объекту Command.
- Управлять транзакциями сессии с помощью методов BeginTrans, CommitTrans, RollbackTrans и свойством Attributes, в том числе и вложенными транзакциями, если провайдер предоставляет такую возможность.
- Просматривать ошибки, возвращаемые источником данных, которые сохраняются в коллекции Errors.
- Узнать версию ядра ADO с помощью свойства Version.
- Получить информацию о базе данных с помощью метода OpenSchema.
- Использовать такую возможность некоторых провайдеров, как вызов объектом Connection именованных команд и хранимых процедур, точно так же, как если бы они были методами этого объекта.

Объект Connection можно использовать независимо от любых прочих объявленных объектов. Чаще всего используются свойство ConnectionString и метод Execute. ConnectionString можно задать несколькими способами. Можно установить свойство ConnectionString, а можно указать строку подключения непосредственно при открытии подключения, в виде аргумента метода Open.

В ConnectionString указывается информация, используемая при установлении соединения с источником данных. В общем случае в ConnectionString указаны параметры в виде: **<имя_параметра> = <значение_параметра>**, разделенных точкой с запятой. ADO поддерживает пять параметров для ConnectionString. Но в зависимости от провайдера, ему могут передаваться и другие параметры, которые не обрабатываются средствами ADO и

передаются непосредственно провайдеру. Вот список, общих для всех провайдеров, параметров, поддерживаемых ADO:
Provider - имя провайдера, который будет использован для подключения к источнику данных.
File Name - имя файла, который будет использован провайдером и в котором содержится предустановленная информация о соединении.
Remote Provider - имя провайдера, который будет использован для открытия соединения на стороне клиента (применимо только для RDS, Remote Data Service).
Remote Server - имя сервера (путь), используемое для открытия соединения на стороне клиента. Используется только при работе со службой RDS.
URL - адрес, идентифицирующий такие ресурсы, как файл или каталог и указывающий, что строка подключения будет взята с указанного URL.
Примечание. После того, как вы установите ConnectionString и откроете подключение, провайдер может изменить значение параметров - определенные в ADO параметры будут переназначены в их эквиваленты для провайдера.
Свойство ConnectionString автоматически наследует значения соответствующего аргумента метода Open, если он был указан. Вы не можете указывать аргументы Provider и File Name одновременно, так как указание File Name означает, что ADO автоматически загрузит соответствующего провайдера. Значение ConnectionString не может меняться, когда соединение открыто. Если в ConnectionString один параметр указан несколько раз, то будет использовано последнее указанное значение.
Синтаксис ConnectionString для основных провайдеров приводится в конце пособия. Более полную информацию см. в MSDN и на сайте ConnectionStrings.

Объект **Command** позволяет задать команду провайдеру на выполнение той или иной операции над данными источника. Он формирует запросы на выборку записей из источников данных, учитывая заданные пользователем параметры. Как правило, выбранные записи возвращаются в объекте Recordset. Объект Command создается на базе таблицы БД или результатов SQL запроса.
Используя методы и свойства этого объекта, можно выполнить следующее:

- Задать описание команды, используя свойства CommandText и CommandStream. Второе из этих свойств позволяет, например, задавать XML-запросы.
- Связать объект Command с открытым соединением, используя свойство ActiveConnection.
- Вызвать команду на исполнение, используя метод Execute объекта Command, или для именованного объекта вызвать команду как метод объекта Connection.
- Формировать объект Recordset как результат выполнения команды.
- В момент вызова команды передавать параметры хранимой процедуре или параметризованному запросу.
- Формировать, при необходимости, объекты Parameter и создавать коллекцию Parameters.

- Управлять эффективностью выполнения команды, используя свойства Prepared и CommandType

Объект **Recordset** представляет набор записей, полученных в результате выполнения операций над источником данных. Он является основным объектом, позволяющим программисту работать с данными, извлеченными и помещаемыми в базу данных. Объект Recordset обеспечивает доступ к записям, выбранным SQL запросом, Его применяют для редактирования записей в источнике данных. Он создается разными способами: методом Execute объектов Connection и Command, либо методом Open объекта Recordset.

В каждый текущий момент можно работать только с одной записью набора - текущей, на которую указывает курсор. Методы и свойства объекта позволяют перемещать курсор по набору записей, производить фильтрацию записей, их сортировку, находить нужную запись. Можно изменять значения отдельных полей некоторых записей, добавлять новые и удалять записи из набора. Измененный набор можно сохранить в базе данных. При этом изменения можно производить оперативно - для каждой записи, подвергшейся изменению, или в пакетном режиме.

Во многом возможность выполнения операций над данными определяется типом и положением курсора, которые устанавливаются свойствами объекта Recordset. В ADO определены четыре типа курсора и два возможных положения, указывающие, где выполняются операции - на клиентской или серверной стороне (**более подробно о курсорах см. ниже - "Дополнительные сведения"**).

Свою специфику, определяющую возможность выполнения отдельных операций, накладывает используемый провайдер. Свойство Supports позволяет выяснить, поддерживает ли провайдер те или иные свойства выбранного курсора.

Свойство Fields является свойством по умолчанию объекта Recordset. Оно возвращает одноименную коллекцию **Fields** полей, элементами которой являются объекты **Field**, каждый из которых позволяет работать с отдельным полем записи.

Объект **Record** похож на объект Recordset, но содержит только одну запись, которая может быть записью набора данных или другим типом объекта (файлом, папкой), в зависимости от провайдера. Но дело в том, что объект Recordset не имеет, ни свойства Records, возвращающего коллекцию записей, ни свойства Item, возвращающего отдельную запись. Поэтому получить явным способом этот объект из набора записей не удается. Некоторые провайдеры возвращают объект Record в тех случаях, когда результатом запроса является объект Recordset, состоящий из одной записи. Обычно он создается методом Open.

Объект Record хотя и подобен набору записей, состоящему из одной записи, но имеет, все-таки, отличающийся набор свойств и методов. Этот объект часто используют при доставке данных из Интернета, поскольку его

контентом могут быть файл или каталог. В роли провайдера часто выступает Microsoft OLE DB Provider for Internet Publishing. Этот провайдер позволяет использовать объект Record для управления данными, такими как каталоги и папки файловой системы, а также сообщениями в системе электронной почты. Источником данных в этом случае выступает абсолютный или относительный адрес URL, сочетаемый с объектом Connection.

Объект **Stream -** специальный объект, представляющий поток текстовых или бинарных данных. Например, XML-документ может загружаться в поток из командной строки или возвращаться, как результат запроса. С помощью этого объекта можно возвращать изображения, хранящиеся в базе данных (для web-приложений). Объект Stream может использоваться для управления полями или записями, содержащими потоки данных.

Коллекция **Properties** и объект **Propert**y.Многие из объектов ADO имеют свойство Properties, возвращающее одноименную коллекцию Properties объектов Property. Поскольку каждый объект обладает набором свойств, то возникает естественный вопрос: зачем необходимо еще и свойство Properties, которое тоже задает свойства. Дело в том, что свойства разделяются на встроенные и динамические. Встроенные свойства - атрибут объектов ADO и доступны для любого объекта.

Динамические свойства зависят от специфики провайдера, они то и составляют коллекцию Properties. Каждый объект этой коллекции описывает то или иное динамическое свойство объекта ADO, зависящее от специфики провайдера и доступное, естественно, только тогда, когда объект открыт. Иначе, объект Property является контейнером для динамических свойств, определяемых провайдером. Добавлять или удалять элементы коллекции Properties - невозможно, это - прерогатива провайдера.

Использование объектов прямого доступа к данным

Data **A**ccess **O**bjects - DAO.

Задание.

►Разработать и отладить **процедуры создания** каждого из **пяти** типов объектов Recordset, в зависимости от типа открываемого сеанса (рабочей области). Источник внешних данных выбирается самостоятельно.

►В соответствии с целью создания результирующего набора записей, подготовить **процедуры работы** (модификация - добавление, удаление, изменение; поиск; фильтрация; сортировка; транзакции и т.п.) **с записями** в наборах.

Количественный состав процедур **зависит** от типа объекта Recordset и определяется его функциональностью. Результаты выполнения операций –

визуализировать с помощью **графического интерфейса** (пользовательские интерактивные формы).

►Получить **временные оценки** выполнения **однотипных** операций над **соответствующими** наборами записей для анализа производительности в **разных** рабочих областях - MS Jet и ODBCDirect. Построить **диаграммы** зависимости времени выполнения операций от типа набора записей для каждой рабочей области. Выполнить **анализ** и сделать **выводы** о предпочтительности использования типов объектов прямого доступа в каждой конкретной конфигурации доступа к источнику внешних данных.

►В качестве источников данных можно использовать имеющиеся разработки баз данных, выполненные в лабораторном практикуме осеннего семестра. Основные таблицы в базе данных, для адекватности временных оценок, должны содержать **не менее** 50 записей.

►При выполнении задания следует ориентироваться на приведенную ниже последовательность лабораторных занятий.

►Форма отчетности – стандартная**:** именованная (фамилией студента) папка, содержащая файл/файлы с исполняемым кодом; текстовый файл с отчетом. (Например: Иванов \ ЛР1DAO.mdb, ЛР1DAO-Отчет.doc).

Последовательность выполнения задания для технологии DAO

ЛР.1DAO. Работа с объектом Recordset типа **таблицы.**

ЛР.2DAO. Работа с объектом Recordset типа **динамического набора записей.**

ЛР.3DAO. Работа с объектом Recordset типа **статического набора записей** и **статического набора записей с последовательным доступом.**

ЛР.4DAO. Работа с объектом Recordset **динамического типа**.

ЛР.5DAO. Оценка производительности работы с объектами DAO.

Дополнительные сведения

Семейство Recordsets. При работе с объектами доступа к данным почти все операции выполняются с помощью объектов Recordset. Семейство Recordsets содержит все открытые объекты Recordset из объекта Database.

Новый объект Recordset автоматически добавляется в семейство Recordsets при открытии объекта и автоматически удаляется из семейства при закрытии объекта.

В приложении допускается определение произвольного числа объектных переменных, представляющих объект Recordset. Разные объекты Recordset могут иметь доступ к одним таблицам, запросам и полям без возникновения конфликтов.

При ссылках на объект Recordset используют его порядковый номер в семействе или значение свойства Name (имя):

Recordsets(0)
Recordsets("имя")
Recordsets![имя]

Примечание. Для одного источника данных или базы данных допускается открытие нескольких объектов Recordset, что может привести к образованию повторяющихся имен в семействе Recordsets. В этом случае необходимо присваивать объекты Recordset объектным переменным и ссылаться на них по именам переменных.

Пример.

В следующем фрагменте VBA-кода используется метод OpenRecordset для открытия пяти объектов Recordset и отображения их содержимого. Для выполнения указанных операций используются процедуры **OpenRecordsetX** и **OpenRecordsetOutput**.

```
Sub OpenRecordsetX()
        Dim wrkJet As Workspace
        Dim wrkODBC As Workspace
        Dim dbsNorthwind As Database
        Dim conPubs As Connection
        Dim rstTemp As Recordset
        Dim rstTemp2 As Recordset

        ' Открывает сеансы Microsoft Jet и ODBCDirect, базу данных
        ' Microsoft Jet и подключение ODBCDirect.
        Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)
        Set wrkODBC = CreateWorkspace("", "admin", "", dbUseODBC)
        Set dbsNorthwind = wrkJet.OpenDatabase("Борей.mdb")
        Set conPubs = wrkODBC.OpenConnection("", , , _
                "ODBC;DATABASE=Pubs;UID=sa;PWD=;DSN=Publishers")
        ' Открывает пять объектов Recordset и отображает
        ' содержимое каждого объекта.
        Debug.Print "Статический с последовательным доступом, " & _
                "где источником является объект QueryDef..."
        Set rstTemp = dbsNorthwind.OpenRecordset( _
                "Десять самых дорогих товаров", dbOpenForwardOnly)
        OpenRecordsetOutput   rstTemp

        Debug.Print "Динамический набор записей только для чтения, " & _
        "где источником является инструкция SQL..."
        Set rstTemp = dbsNorthwind.OpenRecordset( _
        "SELECT * FROM Сотрудники", dbOpenDynaset, dbReadOnly)
        OpenRecordsetOutput   rstTemp
```

```
        ' Используется свойство Filter для отбора записей
        ' при следующем вызове метода OpenRecordset.
        Debug.Print "Отбор записей из существующего " & _
                "объекта Recordset..."
        rstTemp.Filter = "Фамилия >= 'Ж'"
        Set rstTemp2 = rstTemp.OpenRecordset()
        OpenRecordsetOutput   rstTemp2

        Debug.Print "Набор записей динамического типа " & _
                "для подключения ODBC..."
        Set rstTemp = conPubs.OpenRecordset( _
                "SELECT * FROM stores", dbOpenDynamic)
        OpenRecordsetOutput   rstTemp

        ' С помощью свойства StillExecuting проверяет, готов ли
        ' объект Recordset для обработки.
        Debug.Print "Статический набор записей на основе " & _
                "асинхронного запроса для подключения ODBC..."
        Set rstTemp = conPubs.OpenRecordset("publishers", _
                dbOpenSnapshot, dbRunAsync)
        Do While rstTemp.StillExecuting
                Debug.Print "    [еще выполняется...]"
        Loop
        OpenRecordsetOutput   rstTemp

        rstTemp.Close
        dbsNorthwind.Close
        conPubs.Close
        wrkJet.Close
        wrkODBC.Close
End Sub

Sub OpenRecordsetOutput (rstOutput As Recordset)
        ' Отображает указанный объект Recordset.
        With rstOutput
                Do While Not .EOF
                        Debug.Print , .Fields(0), .Fields(1)
                        .MoveNext
                Loop
        End With
End Sub
```

Объект Recordset. Объект Recordset представляет набор записей в основной таблице или набор записей, который получается в результате выполнения запроса.

Объекты Recordset используются для обработки данных в базе данных на уровне записи. При работе с объектами доступа к данным (DAO) почти все операции выполняются с помощью объектов Recordset. Каждый объект Recordset состоит из **записей** (строк) и **полей** (столбцов). Существуют объекты Recordset **пяти** типов:

● **Объект Recordset типа** таблица (**Table-type Recordset)** — программное представление основной таблицы; используется для добавления, изменения или удаления записей из отдельной таблицы базы данных (**только** в рабочей области **Microsoft Jet**). Точного аналога в ADO **не имеет**.

● **Объект Recordset типа** динамический набор записей **(Dynaset-type Recordset)** — набор обновляемых записей, полученный в результате выполнения запроса. Объект Recordset типа динамического набора записей позволяет добавлять, изменять или удалять записи в основной таблице или таблицах. В объекте Recordset этого типа могут содержаться поля из одной или нескольких таблиц базы данных. В ADO данный тип объекта соответствует объекту Recordset, созданному с типом курсора **keyset cursor.**

● **Объект Recordset типа** статический набор записей (**Snapshot-type Recordset)** — статическая копия набора записей, используемая для поиска данных или создания отчетов. Объект Recordset типа статического набора записей может содержать поля из одной или нескольких таблиц базы данных, но не допускает обновления полей. Этот тип набора данных соответствует в ADO объекту Recordset, созданному с типом курсора **static cursor**.

● **Объект Recordset типа** статический набор записей с последовательным доступом (**Forward-only-type Recordset)** — аналогичен статическому набору записей с тем лишь исключением, что в нем отсутствует указатель. Пользователь имеет возможность прокручивать записи только вперед, без возможности применения методов Move. Это повышает быстродействие в ситуациях, когда достаточен однократный проход по набору записей. Этот тип набора данных соответствует в ADO объекту Recordset, созданному с типом курсора **forward-only cursor.**

● **Объект Recordset** динамического типа (**Dynamic-type Recordset)** —набор записей запроса по одной или нескольким основным таблицам, в котором допускается добавление, изменение или удаление записей в результирующем наборе (а не в исходных таблицах, в отличие от Dynaset) (**только** в рабочей области **ODBCDirect**). В таком объекте Recordset отображаются записи, добавляемые, удаляемые или изменяемые в основных таблицах другими пользователями. Этот тип набора данных соответствует в ADO объекту Recordset, созданному с типом курсора **dynamic cursor**.

При создании нового объекта Recordset тип объекта определяется с помощью **аргумента** *type* метода OpenRecordset:

Set *recordset* = *object***.OpenRecordset (***source***,** *type***,** *options, lockedits***)**

Значениями аргумента *type* может быть одна из констант: dbOpenTable, dbOpenDynaset, dbOpenSnapshot, dbOpenForwardOnly, dbOpenDynamic.

В рабочей области Microsoft Jet, если аргумент **тип не задан**, механизм DAO пытается создать тип объекта Recordset с максимально возможными функциональными характеристиками, начиная с типа таблицы. Если таблицу создать не удается, делается попытка создать динамический набор записей,

затем статический набор записей и наконец, набор записей с последовательным доступом.

В рабочей области ODBCDirect, если аргумент тип не задан, механизм DAO пытается создать тип объекта Recordset, обеспечивающий максимально быстрое выполнение запроса, начиная с набора записей с последовательным доступом. Если объект этого типа создать невозможно, механизм DAO делает попытку создать статический набор записей, затем динамический набор записей, и наконец, объект Recordset динамического типа.

При создании объекта Recordset на основе неприсоединенного объекта TableDef в рабочей области Microsoft Jet создаются объекты Recordset типа таблицы. На основе присоединенных таблиц или таблиц в базах данных ODBC, подключенных к ядру Microsoft Jet, могут быть созданы только объекты Recordset типа динамического набора записей или статического набора записей.

Примечание. При описании в программе объекта Recordset и содержащего его объекта Database с помощью объектных переменных необходимо обеспечить, чтобы эти переменные имели одинаковую область определения или одинаковое время жизни. Например, при описании общей переменной, представляющей объект Recordset, необходимо обеспечить, чтобы переменная, представляющая объект Database, содержащий объект Recordset, также была общей, либо описать эту переменную в процедуре Sub или Function с ключевым словом Static.

Объекты Recordset типа динамического набора записей, статического набора записей и статического набора записей с последовательным доступом сохраняются в локальной оперативной памяти. Если в оперативной памяти не хватает места, ядро базы данных Microsoft Jet сохраняет избыточные данные в каталоге временных файлов на диске. Если и этот объем окажется недостаточным, возникает перехватываемая ошибка.

Если созданный объект Recordset содержит записи, то текущей записью становится первая запись. Если записи в объекте отсутствуют, свойство RecordCount получает значение 0, а свойства **BOF** и **EOF** значение True

Изменить положение указателя текущей записи позволяют методы **MoveNext, MovePrevious, MoveFirst и MoveLast**. Для объектов Recordset типа статического набора записей с последовательным доступом поддерживается только метод MoveNext. При выполнении цикла по всем записям с помощью методов Move ("прохода" по объекту Recordset) свойства BOF и EOF используются для проверки начальной и конечной границы объекта Recordset.

Для объектов Recordset типа динамического и статического набора записей в рабочей области Microsoft Jet допускается также использование методов группы **Find**, таких как FindFirst, позволяющих найти запись, удовлетворяющую определенным условиям. Если запись не обнаруживается, свойство NoMatch получает значение True. Для объектов Recordset типа таблицы допускается сканирование записей с помощью метода **Seek**.

Тип созданного объекта Recordset задается свойством Type, а возможность изменения записей в объекте определяется свойством Updatable.

Сведения о структуре основной таблицы, такие как имя и тип данных каждого поля (объект Field) и индекса (объект Index) сохраняются в объекте TableDef.

ActiveX **D**ata **O**bjects - ADO.

Задание.

►Разработать и отладить **процедуры создания** объектов Recordset для каждого из **четырех** типов курсоров.
►В соответствии с целью создания результирующего набора записей, подготовить **процедуры работы** (модификация - добавление, удаление, изменение; поиск; фильтрация; сортировка; транзакции и т.п.) **с записями** в наборах.
Количественный состав процедур **зависит** от функциональности используемого типа курсора, определяемого, к тому же, поддержкой провайдера. Результаты выполнения операций – **визуализировать** с помощью **графического интерфейса** (пользовательские интерактивные формы).
►Получить **временные оценки** выполнения **однотипных** операций над **соответствующими** наборами записей для анализа производительности для разных провайдеров - **Microsoft OLE DB Provider for ODBC, Microsoft.Jet.OLEDB.4.0**, **Microsoft OLE DB Provider for SQL Server**. Построить **диаграммы** зависимости времени выполнения используемых процедур работы от типа курсора для каждого провайдера. Выполнить **анализ** и сделать **выводы** о предпочтительности использования провайдеров в каждой конкретной конфигурации доступа к источнику внешних данных.
►В качестве источников данных можно использовать имеющиеся разработки баз данных, выполненные в лабораторном практикуме осеннего семестра. Основные таблицы в базе данных, для адекватности результатов выполнения предыдущего задания, должны содержать не менее 50 записей.
►При выполнении задания следует ориентироваться на приведенную ниже последовательность лабораторных занятий.
►Форма отчетности – стандартная**:** именованная (фамилией студента) папка, содержащая файл/файлы с исполняемым кодом; текстовый файл с отчетом.
(Например: Иванов \ ЛР1ADO.adp, ЛР1ADO-Отчет.doc).

Последовательность выполнения задания для технологии ADO

ЛР.1ADO. Работа с объектом Recordset, созданным с типом курсора **keyset cursor**.

ЛР.2ADO. Работа с объектом Recordset, созданным с типом курсора **static cursor**.

ЛР.3ADO. Работа с объектом Recordset, созданным с типом курсора **forward-only cursor.**

ЛР.4ADO. Работа с объектом Recordset, созданным с типом курсора **dynamic cursor**.

ЛР.5ADO. Оценка производительности работы с объектами ADO.

Дополнительные сведения

Курсор. Это - важное понятие в объектной модели ADO при работе с базами данных. Курсор - это элемент базы данных, позволяющий управлять перемещением по записям, обновлением данных, видимостью изменений, сделанных другими пользователями.

В реляционных базах данных, в результате запроса возвращается набор строк таблицы (записей). Приложению, работающему с этим набором в каждый текущий момент необходим не весь набор, а отдельная запись или небольшой блок из записей набора. Для работы с набором записей приложению необходим программный механизм, который будет управлять позициями записей в наборе при его изменении, скроллингом - перемещением вперед и назад по записям набора, разрешением конфликтов при одновременном доступе многих пользователей к одной и той же записи. Все эти службы и предоставляются совокупностью программных компонент, называемых курсорами. Они реализованы в виде библиотеки курсоров, являющейся, обычно, частью базы данных. Название "Курсор" связано с тем, что, так или иначе, курсор указывает на текущую запись в наборе.

Поскольку от решения вопросов, относящихся к курсорам, зависит эффективность работы с базой данных, и эти решения могут играть определяющую роль, то следует уметь правильно выбирать тип курсора, положение курсора и другие его характеристики. Тип курсора задает, как будет идти скроллинг, динамику изменения набора записей и многое другое. Положение курсора определяет, где будет идти основная работа с ним - на клиентской или серверной стороне. Рассмотрим возможные **типы курсоров**. Их **четыре**:

- **Forward Only** - допускает перемещение по записям только вперед, не допуская полноценный скроллинг. Он обычно используется тогда, когда необходим только один проход по записям. Его можно использовать и в случаях нескольких проходов, закрывая и заново открывая курсор. После обработки очередной строки освобождаются ресурсы, связанные с ее хранением. По умолчанию этот курсор является динамическим, что означает, что он следит за всеми изменениями, сделанными другими пользователями в показываемых

записях. Конечно, это не относится к уже просмотренным записям. Константа **adOpenForwardOnly** из перечисления CursorTypeEnum задает этот тип курсора.

- **Static** - курсор этого типа, в отличие от динамического, не следит за изменениями, сделанными другими пользователями. Он сохраняет состояние набора записей на момент открытия. В зависимости от реализации статические курсоры могут допускать только чтение записей или возможность их обновления, могут допускать скроллинг или только перемещение вперед. Как правило, статический курсор допускает видимость изменений, сделанных самим приложением. Этот вид курсора обычно применяется, когда необходим скроллинг, но нет необходимости следить
  за изменениями, сделанными другими пользователями. Константа **adOpenStatic** задает этот тип курсора.
- **Keyset** - курсор этого типа является, обычно, альтернативой курсору Forward Only. Он применяется, когда идет интенсивная работа с отдельными записями набора в произвольном порядке доступа к ним. Этот курсор называется курсором, управляемым набором ключей (keyset driven cursor). Для каждой строки из набора создается ключ, обеспечивающий быстрый доступ к любой строке набора.

Что касается наблюдения за изменениями в наборе, то этот вид курсора обеспечивает стратегию, промежуточную между статическим и динамическим курсором. Он позволяет проследить за изменениями значений записей, сделанных другими пользователями, и этим он похож не динамический курсор. Но он не позволяет проследить за изменениями в составе набора - добавлению или удалению строк, изменения порядка их следования. При создании ключей состав набора замораживается. Когда запись удаляется из набора, то, поскольку ключ для нее сохраняется, то такая запись будет видна, как пустая запись - "дыра" в наборе. Добавляемые записи будут видны в виде добавлений в конец набора. Константа **adOpenKeyset** задает этот тип курсора.

- **Dynamic** - динамический курсор обнаруживает все изменения, происходящие с набором записей, сделанные как самим приложением, так и всеми параллельно работающими пользователями. Все вставки, обновления или удаления, сделанные всеми пользователями видимы для этого типа курсора. Этот вид курсора выбирается, когда необходимо обеспечить совместную работу, но, нужно понимать, он требует от сервера больших затрат и при большом числе пользователей может существенно замедлить работу с набором данных. Константа **adOpenDynamic** задает этот тип курсора.

Другой важной характеристикой курсора, кроме его типа, является его положение (Cursor Location). Константы **adUseClient** и **adUseServer** перечисления CursorLocationEnum задают **положение курсора**. В зависимости от установленного значения все необходимые ресурсы и сама работа с данными ведется на клиентской или серверной стороне.

Достоинства работы на стороне клиента состоят в быстром отклике на обращение к записям, не требующим выхода в сеть. Когда работа с записями ведется в основном в режиме их чтения, то это положение курсора наиболее предпочтительно. Некоторым недостатком является то, что при больших наборах клиентскому компьютеру требуются большие ресурсы по памяти для хранения данных. Преимущества работы также теряются, когда интенсивно изменяется состав набора - записи удаляются, добавляются, так как эти изменения должны отражаться и на сервере.

Если курсор расположен на серверной стороне, то объем передаваемых данных может быть существенно уменьшен, поскольку вся обработка ведется на сервере и клиенту передается только необходимые ему записи, а не весь набор. С другой стороны при большом числе активных пользователей каждому из них сервер должен выделить ресурсы, что может быть серьезной нагрузкой на сервер, с которой он может и не справиться. Еще одним недостатков курсора на серверной стороне является то, что в отличие от курсоров на клиентской стороне, не поддерживается работа в пакетном режиме (batch cursor), а возможна работа только с единственной записью.

Работу на клиентской и серверной стороне поддерживают разные библиотеки курсоров. При работе на стороне клиента для обеспечения работы с курсором ADO вызывает специальную службу - Microsoft Cursor Service for OLE DB, поддерживающую единую функциональность для различных провайдеров.

Для того чтобы обеспечить целостность данных в СУБД имеются специальные механизмы, позволяющие временно закрыть доступ к записям базы другим пользователям, пока один из пользователей корректирует их содержание. С одной стороны, возможность закрытия доступа совершенно необходима в некоторых ситуациях, например, чтобы не продать один и тот же билет на самолет или поезд разным пассажирам. С другой стороны, когда, например, в интернете сотни тысяч пользователей обращаются одновременно к базе данных, то закрытие данных существенно снижает общую производительность системы. Система ADO позволяет указывать провайдеру, какой тип закрытия данных следует применять при работе с объектами Recordset.

Конкретный провайдер может не поддерживать все типы закрытия, в этом случае он будет поддерживать возможный ближайший тип закрытия. Свойство LockType объекта Recordset позволяет управлять закрытием доступа. На серверной стороне можно задать одно из трех возможных значений этого свойства: **adLockReadOnly**, **adLockOptimistic**, **adLockPessimistic**. На клиентской стороне возможно только одно значение - **adLockBatchOptimistic**.

Провайдеры. Клиентское приложение, использующее объекты ADO, получает необходимые ему данные, обращаясь к провайдеру. Провайдер или поставщик данных - обеспечивает интерфейс между конкретным источником данных и клиентским приложением. Провайдер способен корректно

транслировать вызовы методов ADO в запросы к источнику данных, выполнять эти запросы и передавать результаты клиентскому приложению. Несмотря на многообразие источников данных, многие из них имеют единый интерфейс запросов, что позволяет иметь одного провайдера на все источники данных, экспонирующие единый интерфейс запросов. Для уникальных источников данных может понадобиться разработка собственного провайдера. Но для наиболее известных источников данных провайдеры уже разработаны как Microsoft, так и другими фирмами, поддерживающими работу с источниками данных.

Провайдер ODBC.
Полное имя этого провайдера - **Microsoft OLE DB Provider for ODBC**. Он является провайдером по умолчанию для объектов ADO, так что если не задать аргумент Provider в строке соединения, то связь по умолчанию будет осуществляться с этим провайдером. Этот поставщик данных позволяет связаться со всеми СУБД с интерфейсом ODBC. Все СУБД, поставляемые Microsoft - Microsoft SQL Server, Microsoft Access (Microsoft Jet database engine), Microsoft FoxPro - обладают этим интерфейсом. Но и СУБД других фирм, например Oracle, обладают, как правило, этим интерфейсом, так что провайдер ODBC реально позволяет связаться с любой профессиональной базой данных.
Провайдер ODBC, являясь провайдером по умолчанию, поддерживает все зависящие от провайдера свойства и методы объектов ADO. Он поддерживает транзакции, в том числе и вложенные транзакции. Однако различные СУБД могут обеспечивать различный уровень поддержки транзакций, например, Microsoft Access поддерживает вложенные транзакции на глубину не более пяти уровней.
Для этого провайдера аргумент Provider свойства ConnectionString следует установить как **MSDASQL**. Типичная **строка соединения** имеет вид:

"Provider = MSDASQL; DSN = dsnName; UID = userName; PWD = userPassword;"

Аргумент DSN (Data Source Name), задает имя источника данных. Это имя **должно быть зарегистрировано** в Администраторе источников данных ODBC, доступном из панели управления. На рис.11 показано окно Администратора ODBC, в котором устанавливается DSN для базы данных dbTestingADO (см. http://www.intuit.ru/department/office/vbaexcel/6/).

Провайдер Microsoft Jet.
Этот провайдер позволяет получить доступ к Microsoft Jet базам данных. Типичная **строка соединения** имеет вид:

"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=databaseName;_
User ID =userName; Password=user Password;"

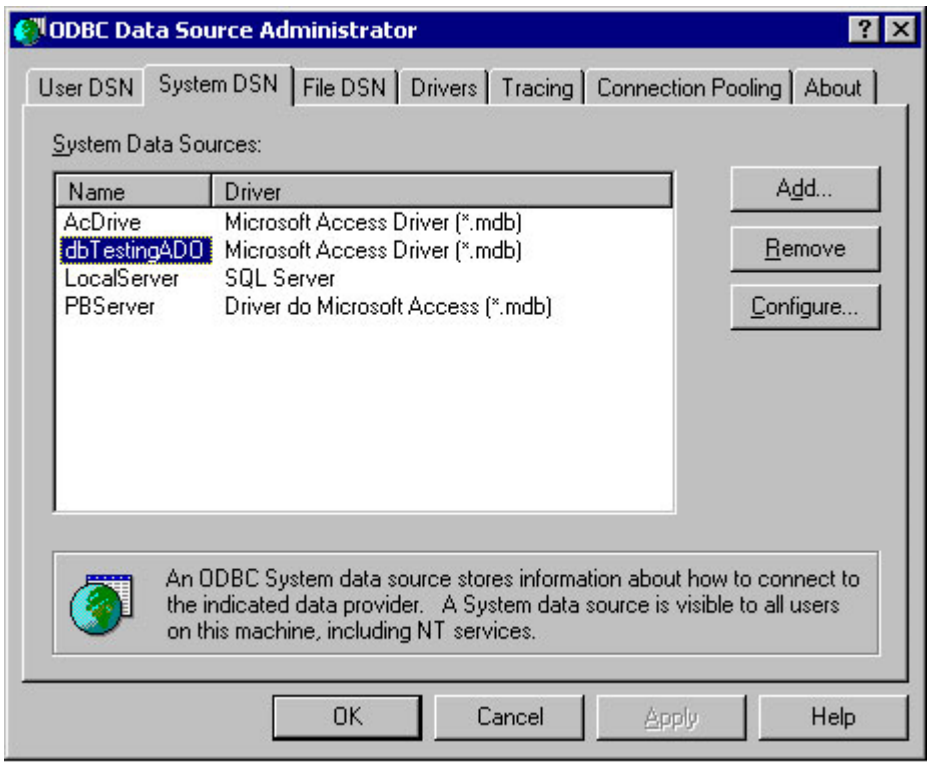

Рис.11.Регистрация источника данных dbTestingADO

В строке соединения провайдеру могут быть переданы и **специфические** свойства. Например, параметр Jet OLEDB: Database Password - позволяет задать пароль к базе данных. Другой способ передачи специфических для провайдера параметров - через коллекцию Propertieis объекта Connection. Большую часть этой коллекции для объектов Connection, Command и Recordset провайдер формирует динамически.

В приведенном ниже **фрагменте** кода **модуля** "TestingADO", иллюстрируется работа с базой данных при поддержке двух вышеописанных провайдеров. Заметьте, во фрагменте отсутствуют описания переменных, объявленных как глобальные в модуле.

```
Public Sub CreateODBCConnect()
        'Создание соединения с тестовой базой данных Access -  "dbTestingADO"
       Dim strConnStr As String
       Dim Start As Single, Finish As Single
       If Con1.State = adStateOpen Then Con1.Close        'закрыть соединение
              'Вариант1: Провайдер ODBC
              Con1.Provider = "MSDASQL"
              strConnStr = "DSN=dbTestingADO; DATABASE =c:\dbPP2010.mdb;"
               'Вариант2: Провайдер Microsoft Jet
              'Con1.Provider = "Microsoft.jet.oledb.4.0"
              'strConnStr = "Data Source=c:\dbPP2010.mdb;"
              Start = Timer
              Con1.Open strConnStr
       'Создать команду
       'задание свойств объекта Command
       Cmd1.ActiveConnection = Con1
```

```
    Cmd1.CommandText = "Select * From [Книги]"
    Cmd1.CommandType = adCmdText
    'Открытие обновляемого объекта Recordset
    With Rst1
        .Open Source:=Cmd1, CursorType:=adOpenDynamic, _
            LockType:=adLockOptimistic

        recExist = False
        .MoveFirst
        Do While Not .EOF
            'Проверка существования записи
            If !Название = "Война и мир" Then recExist = True
            .MoveNext
        Loop
        If Not recExist Then
            .AddNew Array("Автор", "Название", "Год издания", _
                                        "Число страниц", "Цена"), _
                        Array("Лев Толстой", "Война и мир", 2001, 799, 220)
        End If
    End With
    Con1.Close
    Finish = Timer
    Debug.Print "Время работы с таблицей:", Finish - Start
End Sub
```

Обратите внимание на два момента:

- В **варианте2** провайдер Microsoft.jet.oledb.4.0, специально разработан для работы с базой Access. В **варианте1** для работы с этой же базой данных используется **общий** ODBC-провайдер - MSDASQL , позволяющий работать с любыми базами, допускающими интерфейс ODBC. База данных Access, конечно же, позволяет такой способ работы. Однако время работы зависит от выбора провайдера. Этот пример позволяет сравнить два варианта, в каждом из которых устанавливается связь с одним из выше упомянутых провайдеров. Поочередно комментируя операторы, задающие вариант соединения, запускается процедура на выполнение и получаем время ее выполнения для того или иного провайдера.

Временные характеристики, полученные при двух запусках этой процедуры, имеют следующие значения:

Время работы с таблицей: 5,878906
Время работы с таблицей: 0,0390625

Это означает, что производительность работы с ODBC-провайдером для базы данных Access **в данном случае** существенно ниже (на два порядка), чем производительность работы с "родным" провайдером.

- При работе с ODBC-провайдером, помимо DSN задан аргумент DATABASE, указывающий путь к базе данных. Эта информация является дублирующей, поскольку реальный путь к базе данных задается в момент определения DSN.

Тем не менее, это полезно делать, чтобы явно указать, с какой базой данных пользователь намеревается работать.

Провайдер ODBC добавляет в коллекцию Properties объектов Connection, Command, Recordset ряд свойств, часть из которых является специфической для данного провайдера. Некоторые из этих свойств добавляются к не открытым объектам, другие - при открытии объектов. Перечисление всех этих свойств заняло бы слишком много места. Чтобы дать некоторое представление о них, укажем несколько свойств объекта Connection:

- Accesible Tables - булево свойство, указывающее, имеет ли пользователь разрешение на выполнение запросов (SQL-операторов) над таблицами базы данных.
- File Usage - указывает, как драйвер воспринимает источник данных, - как файл или как каталог.
- Special Characters - указывает, какие символы имеют специальный смысл для ODBC драйвера.
- Stored Procedures - определяет доступность использования хранимых процедур.

Приведем еще описания строк подключения для трех стандартных провайдеров.

Провайдер SQL Server.

Полное имя этого провайдера - **Microsoft OLE DB Provider for SQL Server**. Он позволяет получить доступ к данным Microsoft SQL Server. Типичная **строка соединения** имеет вид:

"Provider= SQLOLEDB; Data Source=ServerName; Initial Catalog =databaseName;_
User ID =userName; Password=userPassword;"

Параметр Data Source или Server задает имя сервера. Параметр Initial Catalog или Database - имя базы данных на сервере.

В строке соединения провайдеру могут быть переданы и **специфические** свойства, например, параметр Current Language позволяет задать язык, используемый для системных сообщений. Язык должен быть инсталлирован на SQL Server, иначе возникнет ошибка в момент открытия соединения.

Провайдер Internet Publishing

Полное имя этого провайдера - **Microsoft OLE DB Provider for Internet Publishing**. Он позволяет получить доступ к ресурсам Web-серверов, работающих под управлением Internet Information Server (IIS) . Типичная **строка соединения** имеет вид:

"Provider= MSDAIPP.DSO; Data Source=ResourceURL;
User ID =userName; Password=userPassword;"
или
"URL =ResourceURL; User ID =userName; Password=userPassword;"

Параметр Data Source | URL задает источник данных - URL файла или каталога, расположенного в Web-папке на сервере. Заметьте, во втором случае, когда используется имя URL для второго параметра, первый параметр, определяющий провайдера, задавать не следует. Ошибка возникнет и в том случае, если предварительно будет задано свойство Provider.

Провайдер Oracle.
Полное имя этого провайдера - **Microsoft OLE DB Provider for Oracle**. Он позволяет получить доступ к базе данных Oracle. Типичная **строка соединения** имеет вид:

"Provider= MSDAORA; Data Source=ServerName; User ID =userName; Password=userPassword;"

Параметр Data Source задает имя сервера. В строке соединения провайдеру могут быть переданы и **специфические** свойства, например, параметр OLE DB Services позволяет задать маску, каждый бит которой позволяет включить или выключить соответствующую OLE DB службу. Этот провайдер поддерживает работу только со **статическим** курсором.

Контрольные вопросы

1. Приведите описание существующих способов решения задачи обеспечения доступа к данным. Опишите преимущества и недостатки каждого из них.
2. Перечислите и коротко охарактеризуйте основные (наиболее популярные) известные технологии доступа к данным.
3. Опишите архитектуру концепции доступа к данным фирмы Microsoft. Опишите каждый уровень данной архитектуры, а также роль, которую он играет в общем процессе обеспечения доступа к данным.
4. Опишите объектную модель DAO и основные приемы обеспечения доступа к БД с помощью СУБД Access.
5. Опишите технологию ODBC. Приведите ее особенности, преимущества, недостатки. Опишите архитектуру ODBC.
6. Опишите технологию OLE DB. Приведите ее особенности, преимущества, недостатки. Опишите архитектуру OLE DB, а также механизм взаимодействия компонентов.
7. Опишите технологию ADO. В чем заключается особенность ADO? Перечислите ее преимущества и недостатки. Опишите объектную модель ADO и ее отличия от DAO и RDO.
8. Опишите технологию доступа к данным ADO.NET. В чем ее отличия от ADO? Перечислите основные компоненты этой технологии и опишите механизмы взаимодействия между ними.

Источники информации.

1. Эволюция технологий доступа к данным
http://www.osp.ru/win2000/2003/04/176027/_p2.html
2. Универсальный доступ к данным: Microsoft UDA
http://www.compress.ru/Article.aspx?id=9396
3 Обзор методов доступа к данным
http://alibek09.narod.ru/vb/articles/dba/index.html
4. OLE DB и ADO
http://compress.ru/article.aspx?id=11477&iid=451
5. Универсальный доступ к данным и ADO
http://www.intuit.ru/department/office/vbaexcel/5/
6. ADO.NET: Обзор технологии
http://www.cyberguru.ru/dotnet/ado-net/adonet-overview.html
7. Архитектура ADO.NET
http://msdn.microsoft.com/ru-ru/library/27y4ybxw%28v=VS.90%29.aspx
8. Технологии и средства доступа к реляционным базам данных. ADO .NET
http://club.shelek.ru/viewart.php?id=148