УДК 004.65.3

В.В. Росинский

Метод организации доступа к удаленным источникам данных в корпоративных информационных системах

Проведен анализ проблем, возникающих в процессе разработки и отладки программного обеспечения информационных систем, использующих удаленный доступ к данным. Предложен один из наиболее удобных способов организации этого процесса в информационных системах.

The analysis of problems is conducted which arise in the process of the development and debugging of software of the information systems which use the remote access to information. One of the most comfortable methods of the organization of the access to the remote information in the information systems is suggested.

Проаналізовано проблеми, які виникають у процесі розробки і налагодження програмного забезпечення інформаційних систем які використовують дистанційний доступ до даних. Запропоновано один з найзручніших способів організації цього процесу в інформаційних системах.

Введение. Между узлами корпоративной информационной системы (КИС) происходит обмен большого количества информации, поэтому для ее хранения используют серверы баз данных (БД). Современные системы управления базами данных (СУБД) дают возможность получения аналитических данных, на основе тех, что хранятся в БД, с помощью языка структурных запросов *SQL*. Встроенный в СУБД редактор запросов позволяет составить необходимые условия запроса языком *SQL*, вследствие чего механизм СУБД отображает результаты запроса. Но подобные манипуляции возможны лишь на сервере, где непосредственно размещается БД, что же касается рабочей станции пользователя КИС, то инсталляция, поддержка и обновление БД на ней дороги и не целесообразны. Поэтому удаленное размещение данных КИС, организация доступа к ним и отображение в приемлемом для пользователей формате остается серьезной проблемой для разработчиков и проектировщиков. Данная проблема существенно влияет на цену создания, обслуживание и усовершенствование КИС.

Ключевые слова: базы данных, информационные системы, удаленный доступ, СУБД, *Microsoft SQL Server*, *ADO*, хранимая процедура.

Использование распределенных систем БД в КИС создает условия быстрой и беспрепятственной доставки любой информации от одного узла системы к другому [1]. Основным аргументом в пользу применения распределенных систем БД есть то, что простой перенос принципов, по которым проектируются локальные системы автоматизации, в КИС не возможен, поскольку последние имеют намного более широкую функциональность притом, что их узлы находятся на очень большом расстоянии. Обработка распределенных запросов в этих системах имеет значительные преимущества перед простой обработкой данных на стороне сервера. В распределенных системах БД впервые появился сервер тиражирования данных, более известный как сервер репликации, позволяющий автоматически проводить одновременные изменения в БД нескольких серверов. В распределенных системах БД используют трехзвенную архитектуру, представляющую собой гибкий механизм передачи сообщений между клиентом и сервером и позволяющую организовать взаимодействие между ними многими способами. Но при использовании распределенных систем БД совсем не учитываются нагрузки на трафик, а также удобство внесения корректировок в архитектуру самой КИС. Распределенные систе-

мы БД эффективны лишь при обмене данными между серверами БД и не решают проблем взаимодействия БД и клиентского приложения.

Всемирно известная компания «*Softline Business Intelligence*» специализируется на создании корпоративных хранилищ данных, которые предусматривают реализацию большого числа процессов вытягивания, преобразование и загрузку данных. Эти процессы поддерживаются так называемыми *etl*-инструментами (*extraction*, *transformation*, *loading*). В целом программы *ETL* вытягивают информацию из исходной базы данных, превращают ее в формат, поддерживаемый базой данных назначения, а затем загружают в нее преобразованную информацию. В направлении распределенных систем БД компания достигла значительных успехов, однако *Etl*-Инструменты практически невозможно интегрировать в КИС так, чтобы не усложнить пользователям работу с системой.

Исходя из анализа последних исследований можно утверждать, что существует довольно успешная практика использования распределенных систем БД в КИС в качестве механизма, ответственного за хранение и доставку данных системы к любому ее узлу. С другой стороны, существуют инструменты взаимодействия клиентских приложений с БД, но при этом не существует подхода, который дал бы возможность использовать все преимущества распределенных БД в клиентских программах КИС.

Следует отметить, что при разработке КИС на основе распределенных систем БД, в которых предусмотрен удаленный доступ клиентских программ к БД, возникает множество проблем. Остановимся на основных:

- большое количество технологий, призванных решать проблему организации доступа к удаленным данным, и, как следствие, – проблема выбора;
- удобство использования технологии в связке с платформой или ОС, для которой она была создана;
- сложность разработки приложений на основе технологий, способных сделать его независимым от используемого источника данных (и от используемой СУБД);
- снижение скорости и удобства при разработке и внесении корректировок в структуру КИС;
- нагрузка на сетевой трафик и, как следствие, снижение скорости работы в сети;
- наличие повторяемых блоков;
- снижение уровня безопасности и высокая вероятность повреждения данных.

Исследование технологий доступа к данным

Практическая задача данной статьи – исследовать технологии, предоставляемые разработчиками СУБД для организации отдаленного доступа, и предложить образ использования этих технологий для доступа клиентских программ КИС к распределенным системам БД. Другая задача – исследование механизмов, позволяющих осуществить обработку данных непосредственно на серверах БД, что придает КИС большей централизованности и снижает стоимость ее обслуживания, поскольку в таком случае доставка необходимой аналитики на рабочие станции, задействованные в КИС, должны иметь не только БД, но и средства для работы с ними.

Технология ODBC (*Open Database Connectivity* – открытый интерфейс взаимодействия с базами данных). В большинстве систем проектирования БД приложения основаны на одном их типе. В таких простых схемах разработчик приложения может программировать, напрямую используя системный интерфейс БД. Хотя подобный подход обеспечивает быстрый и эффективный доступ к данным, могут возникать проблемы, когда задача расширяется, и разработчику приходится дорабатывать программу. При данном подходе это означает, что каждая готовая программа должна иметь различные версии с поддержкой всевозможных типов БД. Если компании расширяются или объединяются, приложение должно получить доступ к БД, основанным на различных платформах.

ODBC – одна из первых технологий, предложенная *Microsoft* для унификации доступа к источникам данных. Это программные интерфейсы (*API*) на языке *C* для подключения при-

ложений к различным СУБД. При подключении с помощью *ODBC* приложение становится независимым от используемого источника данных (и от используемой СУБД). Независимость реализуется с помощью промежуточных библиотек, включающих в себя код, специфичный для данной СУБД, и предоставляющих унифицированный интерфейс для *ODBC*-приложений (рис. 1). Такие библиотеки называются *ODBC*-драйверами, и их обычно предоставляют сами разработчики СУБД. Единственное неудобство *ODBC* – это отсутствие объектно-ориентированного подхода [2]. Технология *ODBC* обеспечивает общий интерфейс для доступа к разнородным базам данных стандарта *SQL*. *ODBC* использует язык *SQL* как стандарт для доступа к данным. Этот интерфейс очень удобен: одно приложение может обращаться к различным базам данных *SQL* через общий набор команд.

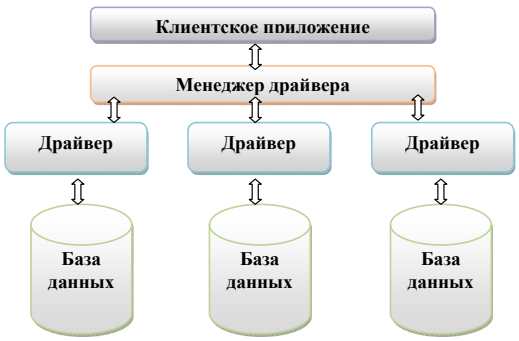

Рис. 1. Архитектура *ODBC*

Технология OLE DB (*Object Linked Embedding*) построена на *ODBC* и расширяет ее до компонентной архитектуры, которая обеспечивает высокоуровневый интерфейс доступа к данным. Эта архитектура предоставляет постоянный доступ к *SQL*-данным, не *SQL*-данным и неструктурированным источникам данных по локальным сетям и *Internet* (например, *Microsoft Exchange Server*, хранилище, которого не содержит реляционные данные). В действительности для доступа к *SQL*-данным *OLE DB* использует *ODBC* как самую подходящую архитектуру для работы с *SQL* [3]. На рис. 2 показано, что *OLE DB* состоит из трех компонентов: потребителя данных (приложения); поставщика (провайдера) данных, содержащего и предоставляющего данные; служебного компонента, обрабатывающего и транспортирующего данные (в частности, процессоры запросов, процессоры курсоров). *OLE DB* – единый *API*, обрабатывающий как совместимые с *SQL* источники данных, так и несовместимые, такие, как почта и каталоги.

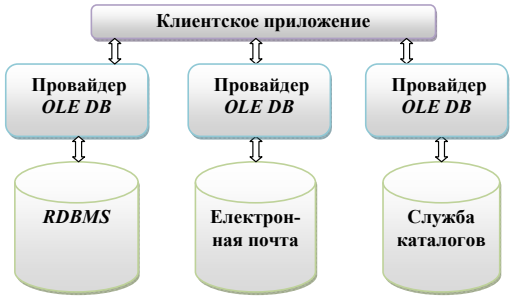

Рис. 2. Компоненты *OLE DB*

OLE DB представляет собой интерфейс системного уровня, обеспечивающий доступ к разным источникам данных, изолируя программу от вида источника. *OLE DB* определяет набор интерфейсов компонентной объектной модели (*Component Object Model – COM*) разных систем, включающих в себя службы управления БД для обеспечения универсального доступа к данным. Суть технологии точно такая же – должны существовать драйверы, через которые осуществляется непосредственное соединение с СУБД, и только через них уже ведет работу с данными прикладная программа. *OLE DB* может работать и через *ODBC*-соединения при помощи специального драйвера, который подключается к *ODBC*-драйверам.

Технологии DAO и RDO. DAO – Data Access Objects (объекты доступа к данным). Базируется на технологии БД *Microsoft Jet* (*JET*) – процессоре БД, предназначенном для *Microsoft Access*. *JET* был первым объектно-ориентированным интерфейсом для связи с *Access*. Приложения, использующие *Access*, могут задействовать *DAO* для прямого доступа к данным. Поскольку *DAO* создавалась вслед за *Access*, применение этой технологии – самый быстрый и наиболее эффективный способ доступа к БД *Access*. *DAO* может работать и с отличными от *Access* БД, такими, как *SQL Server* и *Oracle*. *DAO* использует *ODBC*, но, поскольку метод *DAO* спроектирован специально для взаимодействия

с *JET*, последний транслирует запросы между *DAO* и *ODBC* (рис. 3). Этот дополнительный шаг трансляции и есть причиной замедления работы с БД, отличными от *Access*. Чтобы преодолеть это ограничение, разработчики *Microsoft* создали *RDO* (*Remote Data Objects*). *RDO* обращается к *ODBC API* напрямую, минуя *JET*.

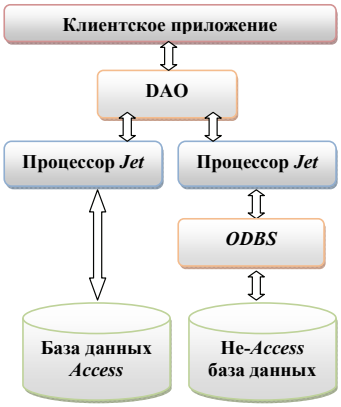

Рис. 3. Использование *DAO* для доступа к БД

Технологии ADO. OLE DB обеспечивает связывание для программистов на *C* и *C++*, а также программистов, использующих языки с *C*-подобными вызовами функций. Такие языки, как *VB* и *VBScript*, не поддерживают тип данных «указатель» (адресных переменных). Следовательно, они не могут использовать связывание в стиле *C* и прямое обращение к *OLE DB*.

ADO работает с объектами *DAO* и *RDO*, а также поддерживает более простые модели, чем *DAO* и *RDO* (хотя с избыточной функциональностью, так что можно выполнить операцию несколькими способами). Объектная иерархия в *ADO* более однородная, чем в *DAO*. *ADO* содержит несколько встроенных объектов, которые упрощают доступ к данным из информационных хранилищ.

На рис. 4 показаны несколько способов, с помощью которых приложение связывается с БД. Например, *VB*-программист может использовать *ADO* для соединения приложения с провайдером *OLE DB*. Если БД не поддерживает *OLE DB*, приложение может задействовать *ODBC*. Программист на *Visual C++* может применять *ADO* или соединяться напрямую через *OLE DB*.

Набор строк (*Recordset*) – центральный объект в *ADO*. Объект *Recordset* представляет собой набор записей (таблицу) и поддерживает типы курсоров *adOpenForwardOnly*, *adOpenKeyset*, *adOpenDynamic* и *adOpenStatic*. Курсор может быть как на стороне сервера (по умолчанию), так и на стороне клиента.

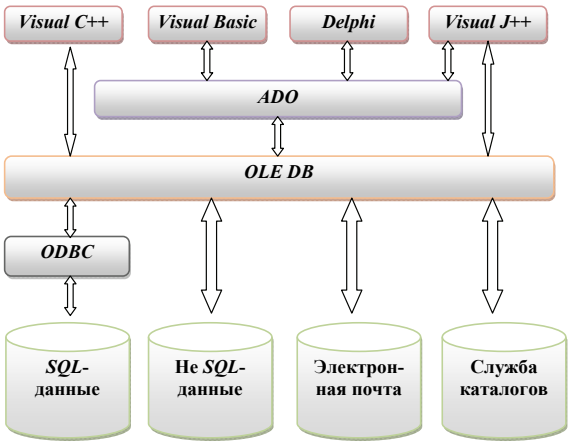

Рис. 4. Использование *ADO* для доступа к БД

Для доступа к записи *ADO* требуется просканировать набор строк последовательно. Для доступа к нескольким таблицам необходимо выполнить запрос на объединение *JOIN*, чтобы получить результат в виде набора строк. Хотя объект *Recordset* поддерживает доступ к данным без соединения с ними, *ADO* изначально был спроектирован для данных, с которыми установлено соединение. Такой метод доступа вынуждает хранить важные ресурсы на стороне сервера. Вдобавок для передачи набора строк следует использовать метод упорядочивания, названный *COM marshalling* – процесс преобразования типов данных, который, естественно, занимает полезные ресурсы системы [4].

Начиная с *ADO* 2.1, *Microsoft* добавляет поддержку *XML* в объектную модель *ADO*, что позволяет хранить набор строк *Recordset* как *XML*-документ. Однако только при появлении *ADO* 2.5 ряд ограничений *XML*, который сохранялся в версии *ADO* 2.1 (например, жесткая иерархия объектов *Recordset*), был устранен. Хотя *ADO* может преобразовать документ *XML* в набор *Recordset*, он в состоянии читать только документы в собственной схеме, известной как *Advanced Data TableGram* (*ADTG*).

В поисках механизма доступа к несвязанным данным *Microsoft* расширяет *ADO* и вво-

дит службу *Remote Data Services* (*RDS*), созданную после *ADO*, и разрешает передачу объекта *Recordset* клиенту (например, в *Web*-браузер) в отсутствие активного соединения. Однако *RDS*, как и *ADO*, использует упорядочивание *COM marshaling* для передачи набора строк от сервера клиенту [5].

Преимущества использования *ADO*

• Большая часть программных средств поддержки этой технологии поставляется в составе ОС, а потому разработчик БД-приложения избавлен от необходимости их внедрения.

• Использование *ADO* позволяет получить доступ к данным, созданным с помощью нетрадиционных технологий, таких как *XML*.

• Компоненты *ADO* допускают асинхронное выполнение операторов *SQL* и позволяют отслеживать процесс выполнения команд с помощью обработчиков событий. Это дает пользователю наглядную информацию о том, насколько далеко продвинулось выполнение запроса.

• В отличие от *BDE*-компонентов механизм *ADO* позволяет остановить работу программ БД без потери информации с помощью *Program Reset*.

Использование *ADO* для удаленного доступа к данным

ADO позволяет получать данные из разных источников (реляционных БД, текстовых файлов и др.) в объектно-ориентированном виде. *ADO* позволяет осуществлять доступ и манипулировать данными с помощью любого провайдера *OLE DB*. *ADO* содержит набор объектов, используемых для соединения с источниками данных, а также для чтения, добавления, восстановления и удаления данных [6].

На рис. 5 представлена объектная модель *ADO*. Объект *ADO Connection* используется для установления связи с источником данных. Он представляет собой единый сеанс взаимодействия с этим источником. С его помощью проводится установка параметров соединения, объект *Connection* позволяет начинать и завершать транзакции. С помощью объекта *Connection* можно выполнять команды-запросы и *Sql*-операторы с помощью метода *Execute*. Если команда возвращает строки, то по умолчанию происходит автоматическое создание объекта *Recordset*. Для получения подробной информации об ошибках, возникающих во время выполнения операции, используется объект *Error*.

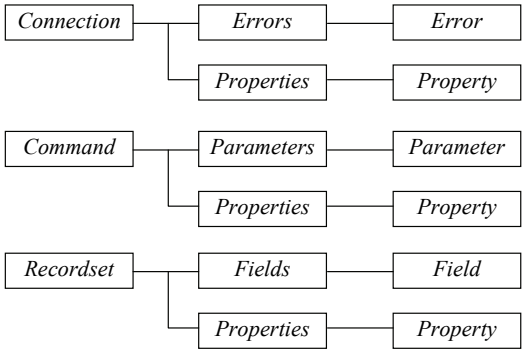

Рис. 5. Объектная модель *ADO*

Объект *Command* содержит команду, применяемую к источнику данных. Команды могут представлять собой простые *Sql*-операторы или вызовы сохраненных процедур [7]. Именно благодаря этому объекту клиентская программа выбирает, какая именно хранимая процедура и с какими параметрами запуститься на сервере БД. Результат выполнения этой процедуры, если такой будет, вернется в клиентскую программу в объекте *Recordset*.

Объект *Recordset* представляет набор записей, полученных из источника данных. Этот объект может применяться для добавления, удаления, восстановления и просмотра (скроллинга) набора записей. Объект *Recodset* может быть открыт непосредственно или создан из объектов *Connection* или *Command*.

Хранимые процедуры

Механизм работы современной КИС построен по принципу клиент–сервер, т.е. программа отсылает на сервер одну или более команд, которые там выполняются. В ответ сервер отправляет клиенту (программе) результат обработки запроса. Этим результатом может быть как сообщение об успешном завершении выполнения команды, которое занимает как всего несколько байт, так и огромный массив данных, который содержит тысячи строк и занимает несколько мегабайт. Клиент может об-

работать полученный результат и на основе своего результата отослать серверу новый запрос. В рассмотренной ситуации логика обработки данных реализована на клиенте. Сервер получает набор инструкций и выполняет их [8]. При написании программного обеспечения разработчик должен позаботиться о разработке запросов, которые корректно работают с данными и выполняют все необходимые действия.

Такой подход к обработке данных имеет несколько недостатков. Например, если нужно изменить логику обработки данных, то приходится менять исходный код программы, после чего заново компилировать ее и распространять среди всех пользователей. Кроме того, если та же логика обработки данных используется в нескольких программах, то в наихудшем случае для каждой из этих программ потребуется повторить процесс разработки запросов, а в лучшем – переносить код из работающей программы.

Как видно из процесса взаимодействия сервера и клиента, алгоритмы обработки данных будут реализованы в виде набора блоков команд, которые поочередно отправляются на сервер. После выполнения блока программа получает определенный результат, после обработки которого решается, какой следующий блок и с какими параметрами будет выполняться. В некоторых ситуациях обмен между клиентом и сервером наборами команд и результатами может занимать много времени и генерировать большой сетевой трафик, что отрицательно отразится на работе программы в целом и на работе других пользователей КИС.

Следует сказать и о безопасности. Для выполнения обработки данных пользователь должен иметь соответствующие права доступа. Предполагается, что эти права будут использованы программой для доступа к данным. Однако нельзя быть до конца уверенным, что пользователь не сможет обратиться к данным прямо, например, с помощью программы *Query Analyzer*, и выполнить неразрешенные действия. Ошибка разработчика при создании запроса может иногда привести к повреждению данных. Кроме того, нельзя не учитывать, что злоумышленник или разработчик способен изменить код запроса для получения несанкционированного доступа к данным или для их повреждения и даже уничтожения [9].

Все сказанное демонстрирует недостатки подхода к разработке систем, когда логика обработки данных реализуется на клиенте. Описанные проблемы могут быть решены путем переноса алгоритмов обработки данных на сервер. В этом случае программа сообщает серверу, который именно набор команд необходимо выполнить. Программой могут быть указанные параметры, которые в зависимости от реализации алгоритма будут влиять на ход выполнения процесса обработки данных. При этом программа сможет получать только конечный результат выполнения. Все промежуточные результаты будут обработаны сервером. Это позволяет снизить сетевой трафик. Этот набор команд, сохраненных на сервере и выполняемых как одно целое, в терминологии *SQL-Server* называется хранимой процедурой (*Stored procedure*).

Использование хранимых процедур позволяет снизить стоимость сопровождения системы и дает возможность избавиться от необходимости менять клиентские приложения. Если понадобится изменить логику обработки данных, чтобы ее стали использовать все программы сети, количество которых может начислять десятки и сотни, то довольно будет изменить только хранимую процедуру.

Кроме того, использование хранимых процедур также позволяет значительно повысить безопасность данных. Программа или пользователь получает лишь специальное право на выполнение сохраненной процедуры, которая и будет обращаться к данным. Доступа же к собственно данным пользователь не получает. В хранимой процедуре можно реализовать проверку на правильность выполняемых изменений, которая обеспечит логическую целостность данных. Также можно реализовать проверки на права пользователя выполнять те или другие действия.

Окончание на стр. 32

Заключение. В статье изложены результаты исследования механизмов, которые позволяют организовать доступ к распределенным системам БД в КИС и оптимизировать разработку, тестирование и внесение изменений в функциональную часть самой КИС. Кроме того, предложена модель использования компонентов технологии *ADO*, которая в комбинации с хранимыми процедурами обеспечивает обработку запросов с целью получения необходимой аналитической информации клиенту и решает проблему получения готовой для отображения в КИС аналитической информации, которая храниться на сервере БД централизованно и на безопасность и целостность которой не влияют пользователи самой системы.

1. *http://citforum.ru/database/kbd*96/45*.shtml*
2. *Станкевич В.* Технологии доступа к данным от *Microsoft* // *Software*. – 2007. – № 11. – С. 48–52.
3. *Вей-Менг Ли*. Эволюция технологий доступа к данным // *SQL Server Magazine*. – 2003. – № 1. – С. 73–81.
4. *http://www.sqlmag.com/article/troubleshooting/Using-Performance-Profiler-to-Troubleshoot-ADO-NET-applications.aspx*
5. *Сивакумар Харинатх*, *Стивен Куинн*. *SQL Server*2005 *Analysis Services* и *MDX* для профессионалов. – К.: Диалектика, 2008. – С. 451–452.
6. *Шпеник М.*, *Следж О.* Руководство администратора баз данных *Microsoft SQL Server* 2000. – М.: Вильямс, 2004. – С. 328–331.
7. *Нойес Б.* Привязка данных в *Windows Forms*. – М.: Бином-Пресс, 2009. – С. 255–259.
8. *Brian Paulen*, *Jeff Finken*. Pro SQL Server 2008 Analytics: Delivering Sales and Marketing Dashboards.– NY: Apress, 2009. – P. 234–236.
9. *Кандзюба С.П.*, *Громов В.Н. Delphi* 6/7. Базы данных и приложения. – К.: ДиаСофт, 2002. – С. 310–312.

Поступила 15.11.2010
Тел. для справок: (044) 526-6439 (Киев)
E-mail: *harlam@ukr.net*