ОСНОВЫ РАЗРАБОТКИ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ (2 ЧАСА)

ПРЕПОДАВАТЕЛЬ: БЛЮМ В.С.

ДИСЦИПЛИНА: МДК.02.02 ТЕХНОЛОГИЯ РАЗРАБОТКИ И ЗАЩИТЫ БАЗ ДАННЫХ

ГРУППЫ: ПКС 3.1-17, ПКС 3.2.-17, ПКС 2.5-18

ЦЕЛЬ

1. Познакомиться с:
   - основами ODBC;
   - архитектурой ODBC;
   - функциями ODBC API;
   - соотношением стандарта ODBC и стандарта интерфейса уровня вызовов (CLI).
2. Изучить создание источников данных с помощью утилиты ODBC и с использованием ODBC API.

ОСНОВА ODBC

- Интерфейс ODBC (Open Database Connectivity) был разработан фирмой Microsoft как открытый интерфейс доступа к базам данных. Он предоставляет унифицированные средства взаимодействия прикладной программы, называемой клиентом (или приложением-клиентом), с сервером - базой данных.
- В основу интерфейса ODBC были положены спецификация CLI-интерфейса (Call-Level Interface), разработанная X/Open, и ISO/IEC для API баз данных, а также язык SQL (Structured Query Language) как стандарт языка доступа к базам данных.
- Интерфейс ODBC проектировался для поддержки максимальной интероперабельности приложений, которая обеспечивает унифицированный доступ любого приложения, использующего ODBC, к различным источникам данных. Так, если приложение, соответствующее стандарту ODBC и SQL, первоначально разрабатывалось для работы с базой данных Microsoft Access, а затем таблицы этой базы были перенесены в базу данных Microsoft SQL Server или базу данных Oracle, то приложение сможет и дальше обрабатывать эти данные без внесения дополнительных изменений.
- Для взаимодействия с базой данных приложение-клиент вызывает функции интерфейса ODBC, которые реализованы в специальных модулях, называемых ODBC-драйверами. Как правило, ODBC-драйверы - это DLL-библиотеки, при этом одна DLL-библиотека может поддерживать несколько ODBC-драйверов. При установке на компьютер любого SQL-сервера (базы данных, поддерживающей один из стандартов языка SQL, например, SQL-92) автоматически выполняется регистрация в реестре Windows и соответствующего ODBC-драйвера.

АРХИТЕКТУРА ODBC

- Приложение-клиент, выполняющее вызов функций ODBC.
- Менеджер драйверов, загружающий и освобождающий ODBC-драйверы, которые требуются для приложений-клиентов. Менеджер драйверов обрабатывает вызовы ODBC-функций или передает их драйверу.
- ODBC-драйвер, обрабатывающий вызовы SQL-функций, передавая SQL-серверу выполняемый SQL-оператор, а приложению-клиенту - результат выполнения вызванной функции.
- Источник данных, определяемый как конкретная локальная или удаленная база данных.

Основное назначение менеджера драйверов - загрузка драйвера, соответствующего подключаемому источнику данных, и инкапсуляция взаимодействия с различными типами источников данных посредством применения различных ODBC-драйверов.

ODBC-драйверы, принимая вызовы функций, взаимодействуют с приложением-клиентом, выполняя следующие задачи:

- управление коммуникационными протоколами между приложением-клиентом и источником данных;
- управление запросами к СУБД;
- выполнение передачи данных от приложения-клиента в СУБД и из базы данных в приложение-клиент;
- возвращение приложению-клиенту стандартной информации о выполненном вызове ODBC-функции в виде кода возврата;
- поддерживает работу с курсорами и управляет транзакциями.

Приложение-клиент одновременно может устанавливать соединения с несколькими различными источниками данных, используя разные ODBC-драйверы, а также несколько соединений с одним и тем же источником данных, используя один и тот же ODBC-драйвер.

ФУНКЦИИ ODBC API

Все функции ODBC API условно можно разделить на четыре группы:

- основные функции ODBC, обеспечивающие взаимодействие с источником данных;
- функции установки (setup DLL);
- функции инсталляции (installer DLL) ODBC и источников данных;
- функции преобразования данных (translation DLL).

Объявления всех функций и используемых ими типов данных содержатся в заголовочных файлах. Группа основных функций ODBC API разбита на три уровня:

- функции ядра ODBC;
- функции 1 уровня;
- функции 2 уровня.

Каждый ODBC-драйвер специфицируется как драйвер, поддерживающий определенный уровень функций ODBC API.

Прототипы функций ядра ODBC API находятся в файле Sql.h (C/C++, Visual Studio), а прототипы функций 1 и 2 уровней - в файле Sqlext.h.

Применение #define ODBCVER позволяет указать используемую версию (например, #define ODBCVER 0x0351).

Прототипы функций установки и инсталляции находятся в файле odbcinst.h.

СООТНОШЕНИЕ СТАНДАРТА ODBC И СТАНДАРТА ИНТЕРФЕЙСА УРОВНЯ ВЫЗОВОВ (CLI)

Как уже отмечалось выше, открытый интерфейс доступа к базам данных фирмы Microsoft основан на следующих стандартах:

- спецификация X/Open CAE[1] (Specification "*Data Management*: SQL Call-Level Interface (CLI)");
- спецификация ISO[2] /IEC 9075-3:1995 (E) (Call-Level Interface (SQL/CLI)).

В настоящее время фирма Microsoft поддерживает версию 3.x ODBC API. Приложения, написанные на основе спецификации X/Open и ISO CLI, будут правильно работать с ODBC-драйверами версии 3.x или драйверами "согласованного стандарта" в том случае, если они компилируются с заголовочными файлами ODBC версии 3.x и линкуются с ODBC 3.x библиотеками, а доступ к ODBC-драйверу получают через менеджер драйверов ODBC 3.x. Аналогично, что и сами драйверы 3.x, написанные на основе спецификации X/Open и ISO CLI, будут правильно работать с приложениями при соблюдении этих же условий.

Драйвер ODBC 3.x всегда поддерживает все возможности, используемые приложением "согласованного стандарта", а приложение ODBC 3, которое использует только возможности, предоставляемые ISO CLI, и обязательные средства, описываемые X/Open CLI, всегда будет работать с драйвером "согласованного стандарта".

В дополнение к интерфейсу, специфицированному в стандартах ISO/IEC и X/Open CLI, ODBC реализует следующие возможности:

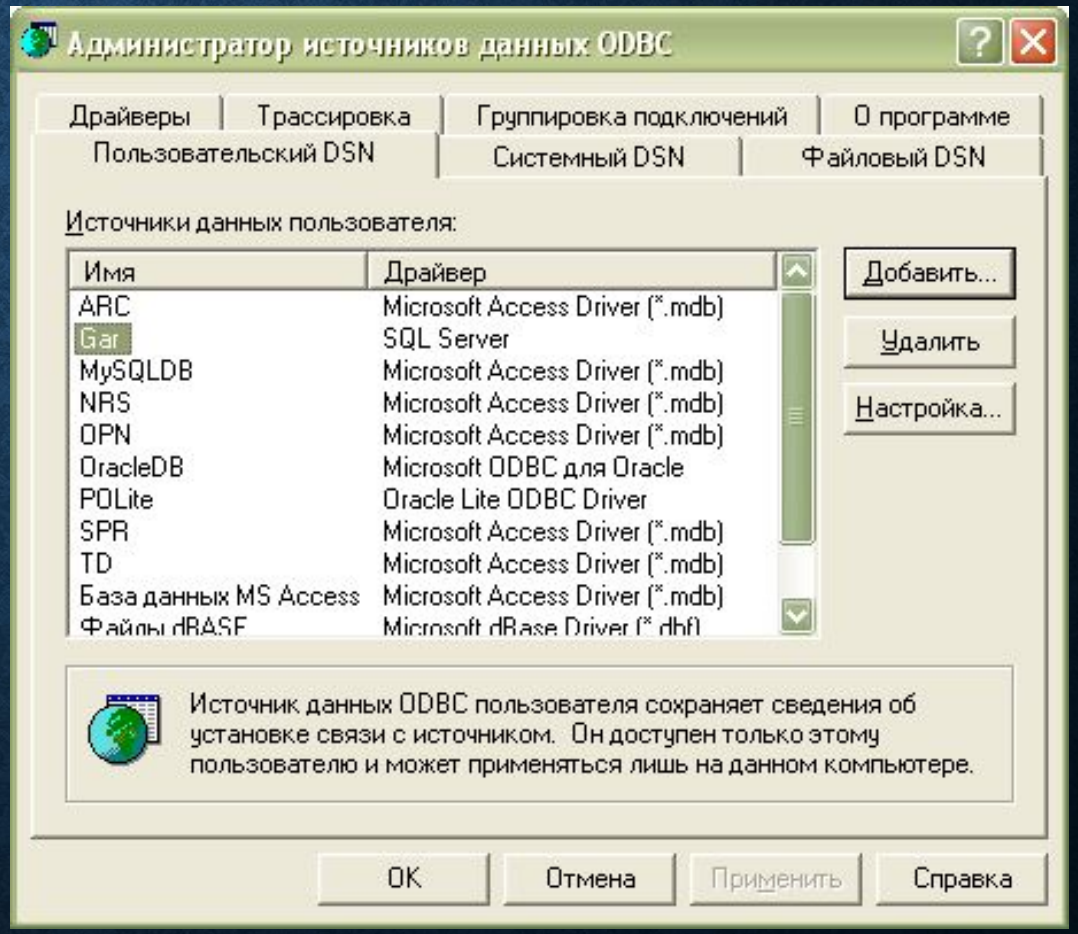
- извлечение нескольких строк (блочная выборка) за один вызов функции;
- связывание с массивом параметров;
- поддержка закладок, включая выборку посредством закладки, закладки переменной длины, блочное обновление и удаление посредством отмеченных операций над непоследовательными строками;
- построчное связывание (row-wise binding);
- связывание со смещением (*binding offsets*);
- поддержка пакетов SQL-операторов как в хранимых процедурах, так и в виде последовательности отдельных SQL-операторов, выполняемых при вызове функций SQLExecute и SQLExecDirect;
- определение точного или приблизительного числа строк курсора;
- применение операции позиционированного обновления и удаления и пакетных удалений и обновлений с использованием функции SQLSetPos;
- поддержка функций каталога, позволяющих получать информацию из схемы базы данных (системных таблиц);
- библиотеки преобразования для кодовых страниц;
- асинхронное выполнение;
- поддержка хранимых процедур, включая escape-последовательности, механизм связывания выходных параметров, функции каталога;
- более продвинутые возможности соединения, включающие поддержку атрибутов соединения и просмотра атрибутов.

СОЗДАНИЕ ИСТОЧНИКА ДАННЫХ

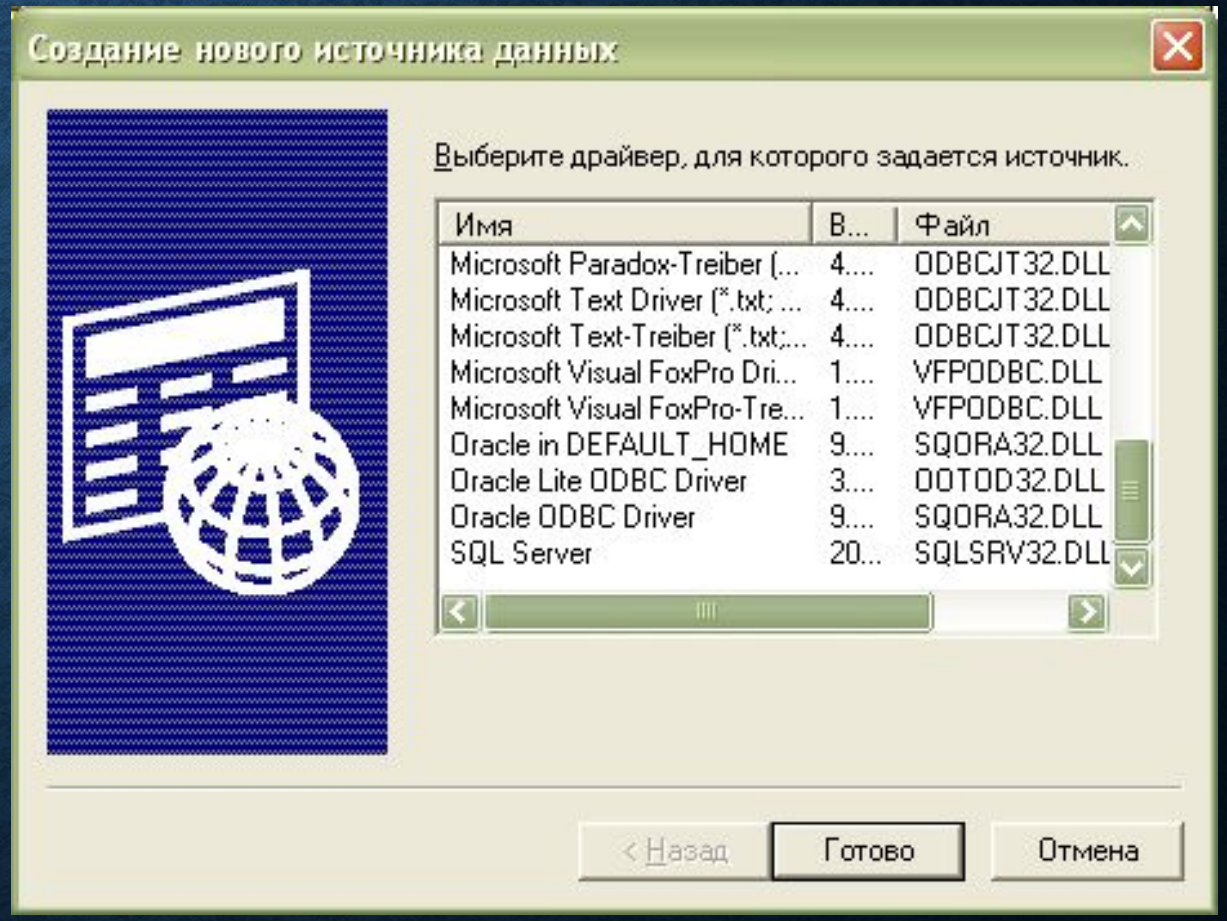
- *Источник данных DSN*, используемый функциями *ODBC API*, первоначально должен быть создан. Это можно выполнить как программно - вызвав функцию *ODBC API*, так и интерактивно - используя утилиту *ODBC* (в зависимости от версии *Windows*, расположенную на панели управления или администрирования).

УТИЛИТА ODBC

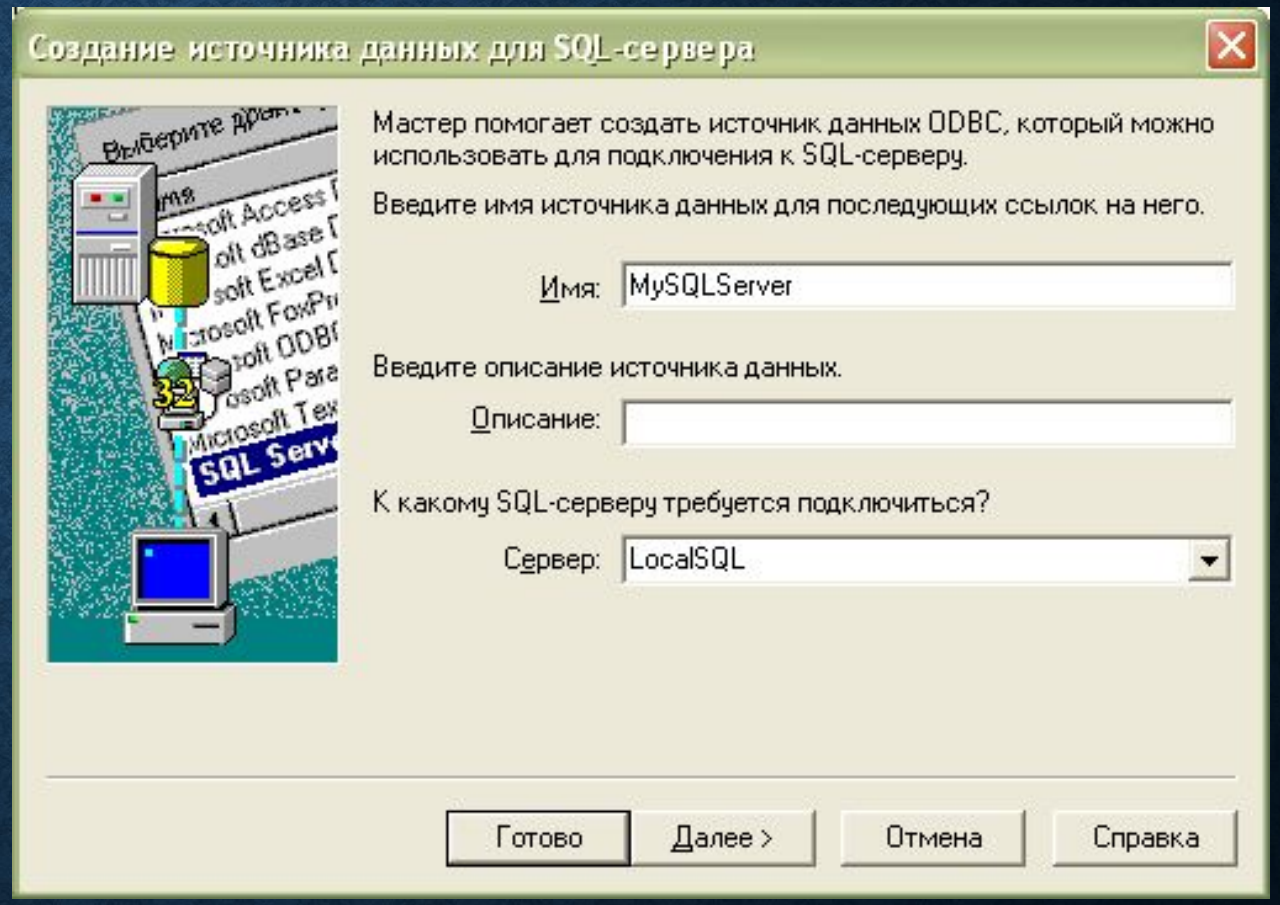
- При использовании утилиты ODBC на вкладке Пользовательский DSN отображается список всех зарегистрированных источников данных.



- При добавлении нового источника данных отображается диалог со всеми зарегистрированными в реестре Windows ODBC-драйверами



- В зависимости от выбранного ODBC-драйвера последовательно отображаются один или несколько диалогов для ввода параметров создаваемого DSN. Так, для создания источника данных, позволяющего работать с базой данных Microsoft SQL Server, следует определить имя создаваемого DSN, имя зарегистрированного SQL-сервера и имя базы данных (на этом сервере), а также ряд дополнительных параметров.



СОЗДАНИЕ ИСТОЧНИКА ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ODBC API

DLL-библиотека ODBCCP32.DLL предоставляет функции ODBC API ConfigDSN и SQLConfigDataSource, позволяющие выполнять регистрацию новых источников данных или удалять информацию об источниках данных из реестра Windows (и из файла ODBC.ini).

Функция ConfigDSN позволяет добавлять, изменять или удалять источники данных и имеет следующее формальное описание:

- BOOL ConfigDSN(
- HWND hwndParent,
- WORD fRequest,
- LPCSTR lpszDriver,
- LPCSTR lpszAttributes);

Для использования в среде Visual Studio функций ConfigDSN и SQLConfigDataSource следует подключить заголовочный файл odbcinst.h.

Параметр hwndParent определяет дескриптор окна или NULL. Если дескриптор не указан, то при выполнении данной функции окно с предложением уточнить параметры не отображается. Параметр fRequest указывает тип запроса, который задается одной из следующих констант:

- ODBC_ADD_DSN - добавление нового источника данных;
- ODBC_CONFIG_DSN - изменение существующего источника данных;
- ODBC_REMOVE_DSN - удаление существующего источника данных.

Параметр lpszDriver содержит описание драйвера, а параметр lpszAttributes - список атрибутов в форме "ключевое слово=значение" (например: DSN=MyDB\0UID=U1\0PWD=P1\0DATABASE=DB1\0\0). Список атрибутов завершается двумя null-байтами.

При успешном завершении функция возвращает значение TRUE, в противном случае вызовом функции **SQLInstallerError** можно получить один из следующих кодов ошибки:

- ODBC_ERROR_INVALID_HWND - ошибка в указании дескриптора окна;
- ODBC_ERROR_INVALID_KEYWORD_VALUE - параметр lpszAttributes содержит ошибки;
- ODBC_ERROR_INVALID_NAME - параметр lpszDriver не найден в системе;
- ODBC_ERROR_INVALID_REQUEST_TYPE - параметр fRequest содержит недопустимое значение;
- ODBC_ERROR_REQUEST_FAILED - нельзя выполнить действие, указанное параметром fRequest ;
- ODBC_ERROR_DRIVER_SPECIFIC - ошибка конкретного драйвера.

Для записи информации об источнике данных в секцию [ODBC Data Sources] секции ODBC.INI реестра Windows функция ConfigDSN вызывает функцию SQLWriteDSNToIni, а для удаления - функцию SQLRemoveDSNFromIni.

Функция SQLConfigDataSource имеет следующее формальное описание:

- BOOL SQLConfigDataSource(
- HWND hwndParent,
- WORD fRequest,
- LPCSTR lpszDriver,
- LPCSTR lpszAttributes);

Параметры функции SQLConfigDataSource аналогичны параметрам функции ConfigDSN, при этом параметр fRequest может принимать следующие значения:

- ODBC_ADD_DSN - добавление нового пользовательского DSN;
- ODBC_CONFIG_DSN - изменение существующего пользовательского DSN;
- ODBC_REMOVE_DSN - удаление существующего пользовательского DSN;
- ODBC_ADD_SYS_DSN - добавление нового системного DSN;
- ODBC_CONFIG_SYS_DSN - изменение существующего системного DSN;
- ODBC_REMOVE_SYS_DSN - удаление существующего системного DSN.

- Функция **ConfigDSN** относится к группе функций установки DLL (setup DLL), а функция **SQLConfigDataSource** - к группе функций инсталляции DLL (Installer DLL).
- При выполнении функция **SQLConfigDataSource** использует значение параметра lpszDriver для получения из системной информации полного пути к Setup DLL конкретного драйвера, загружает эту DLL и вызывает функцию **ConfigDSN**, передавая ей свой список параметров (значение параметра fRequest преобразуется к значению, принимаемому функцией **ConfigDSN**). Перед вызовом **ConfigDSN**, в зависимости от типа обрабатываемого DSN, устанавливается
режим USERDSN_ONLY (пользовательский DSN)
или SYSTEMDSN_ONLY (системный DSN), а перед завершением выполнения функция **SQLConfigDataSource** возвращает режим BOTHDSN.

КОДЫ ВОЗВРАТА

- Все функции *ODBC API* возвращают значения, называемые кодами возврата. *Код возврата* определяет, была ли *функция* выполнена успешно, или характеризует тип произошедшей ошибки.

В ЗАГОЛОВОЧНОМ ФАЙЛЕ SQL.H ОПРЕДЕЛЕНЫ СЛЕДУЮЩИЕ КОДЫ ВОЗВРАТА:

#define SQL_SUCCESS 0	Функция выполнена успешно
#define SQL_SUCCESS_WITH_INFO 1	Функция выполнена успешно, но с уведомительным сообщением
#if (ODBCVER >= 0x0300) #define SQL_NO_DATA 100 #endif	Больше нет строк для извлечения их из результирующего набора. В предыдущей версии ODBC API этот код возврата обозначался как SQL_NO_DATA_FOUND. В версии 3.x код возврата SQL_NO_DATA_FOUND содержатся в заголовочном файле sqlext.h
#define SQL_ERROR (-1)	При выполнении функции произошла ошибка
#define SQL_INVALID_HANDLE (-2)	Указан неверный дескриптор
#define SQL_STILL_EXECUTING 2	Функция, выполняемая асинхронно, пока не завершена
#define SQL_NEED_DATA 99	Для успешного выполнения данной функции следует предварительно определить необходимые данные

Первые два кода возврата определяют, что функция была выполнена, а остальные информируют о типе произошедшей ошибки.

Для определения типа кода возврата в заголовочном файле sqltypes.h введено следующее объявление:

- typedef signed short RETCODE;

ЗАДАНИЕ

- Ознакомиться в вышеописанной теорией и выполните тест расположенный на ресурсе: