ГРАН
СИСТЕМА-С

Обмен информацией с базой данных АСКУЭ

Руководство программиста

2009

Оглавление

1. Основы ODBC API

Open Database Connectivity (ODBC) представляет собой интерфейс прикладной программы (API) для доступа к базам данных. Это основано на спецификациях Call-Level Interface (CLI) от X/Open и ISO/IEC для API баз данных. Как язык доступа к базам данных применяется Structured Query Language (SQL).

ODBC разработан для максимальной способности к взаимодействию, то есть одна прикладная программа может без изменения своего исходного текста работать через интерфейс с какой угодно СУБД. Прикладные программы вызывают функции интерфейса ODBC, которые выполнены в специфических для базы данных модулях, названных драйверами. Использование драйверов изолирует прикладные программы от специфических для базы данных обращений.

Имеются два архитектурных требования:

1. Прикладные программы должны быть способны обратиться ко многим СУБД, используя тот же самый исходный текст без того, чтобы его перетранслировать или заново компоновать.
2. Прикладные программы должны быть способны обратиться ко многим СУБД одновременно (через разные драйверы).

ODBC успешно решает эти проблемы следующим способом:

ODBC является интерфейсом уровня вызовов:

Чтобы решить проблему с тем, как прикладные программы обращаются ко многим СУБД, используя один и тот же исходный текст, существует стандарт CLI. ODBC содержит все функции в спецификации CLI и обеспечивает дополнительные функции, обычно требуемые прикладными программами.

ODBC определяет стандартный синтаксис SQL:

В дополнение к стандартному интерфейсу уровня обращения (вызова), ODBC определяет стандартный синтаксис SQL. Он базируется на спецификации X/Open SQL CAE. Если используемый ODBC синтаксис отличается от того, который применяет конкретная СУБД, производится преобразование на лету. Однако, такие преобразования редки потому, что большинство СУБД уже используют стандартный синтаксис языка SQL.

ODBC предоставляет Driver Manager для управления одновременным доступом к многим СУБД:

Хотя использование драйверов решает проблему одновременного доступа ко многим базам данных, код, необходимый, чтобы сделать это, может быть сложен. Прикладные программы которые разработаны, чтобы работать со всеми драйверами, не могут быть статически связаны с любыми драйверами. Вместо этого они должны загрузить драйверы во время выполнения и вызывать функции в них через таблицу указателей функций. Ситуация становится более сложной, если прикладная программа использует много драйверов сразу. Чтобы избавить программу от проблем с этим, ODBC обеспечивает Driver Manager. Администратор драйверов (Driver

Manager) осуществляет все функции ODBC обычно как вызовы функций ODBC в драйверах и статически связан с прикладной программой или загружен прикладной программой во время выполнения. Таким образом, вызовы из прикладной программы функций ODBC по именам обрабатываются в Driver Manager вместо того, чтобы обращаться по указателю к каждому драйверу. ODBC предоставляет много возможностей СУБД, но не требует, чтобы каждый драйвер поддерживал их все.

Для реализации прикладной программы будем использовать операционную систему WINDOWS, интегрированную среду разработки Microsoft Visual Studio .NET и язык программирования C++.

Использование ODBC в прикладной программы должно осуществляться по алгоритму, изображенному на рис. 1. Основные понятия ODBC будут пояснены ниже.

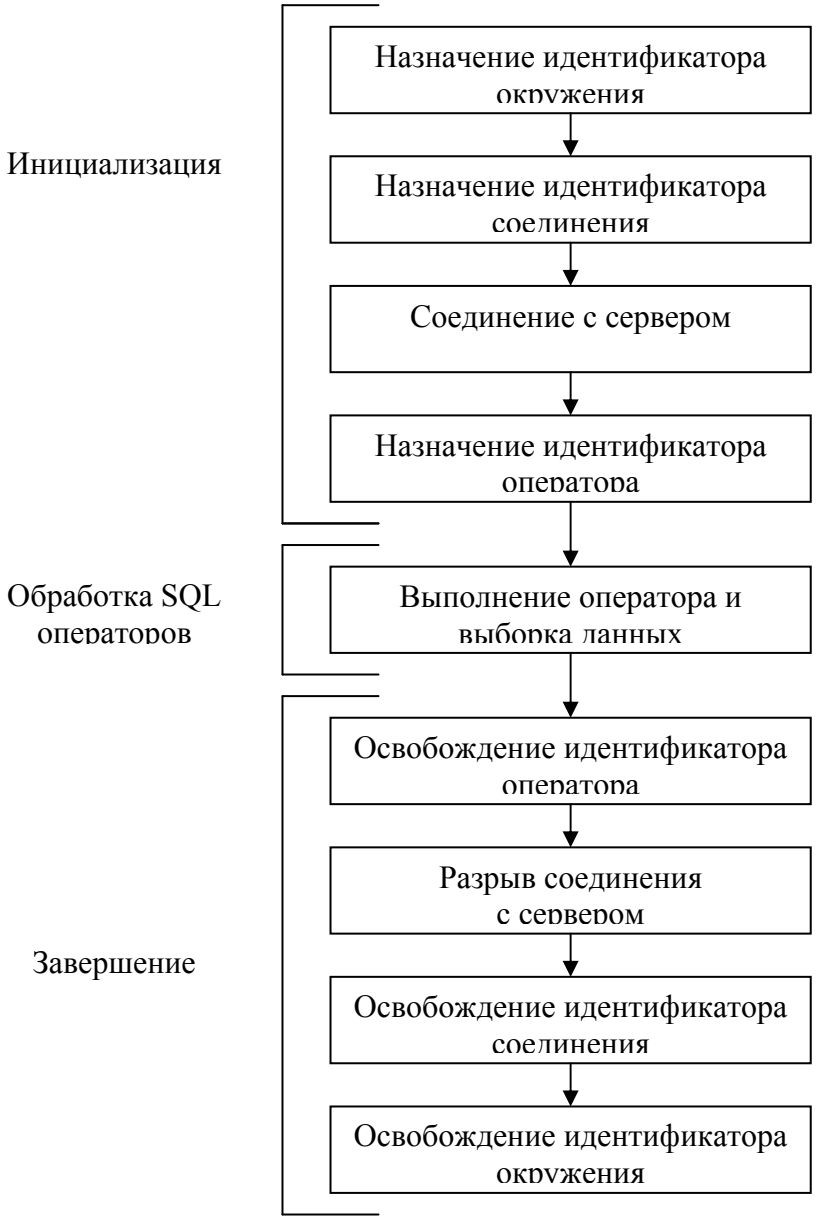

Рис. 1. Основной алгоритм использования ODBC в прикладных программах

Идентификатор окружения – этот идентификатор указывает на область памяти для глобальной информации. Переменная типа HENV также включает в себя любые данные, которые должны быть глобальными для прикладной программы, а именно, сведения обо всех соединениях с базами данных и информацию о том, какое соединение является текущим.

Этот идентификатор указывает на область памяти для информации о конкретном соединении. В то время как каждый идентификатор соединения ассоциируется с единственным идентификатором окружения, этот единственный идентификатор окружения может иметь один или более связанных с ним идентификаторов соединения.

Идентификатор оператора – этот идентификатор, который относится к типу HSTMT, указывает на область памяти для информации о SQL-операторе. Прикладная программа должна запрашивать идентификатор оператора прежде, чем она выдаст SQL-запрос. В то время как каждый идентификатор оператора связывается с единственным идентификатором соединения, каждый идентификатор соединения может иметь один или более связанных с ним идентификаторов операторов.

Соединение с сервером – при управлении выполнением в прикладных программах, как только было назначено окружение, включающее свой идентификатор, могут быть назначены идентификаторы соединения. Аналогично, после назначения идентификатора соединения могут быть назначены идентификаторы операторов. С помощью этих функций устанавливается соединение с сервером базы данных.

Выполнение операторов SQL – существует два способа определения и выполнения SQL-операторов: с предварительной подготовкой и непосредственный. Предварительная подготовка используется в тех случаях, необходимо многократно выполнять прикладную программу SQL-операторов, возможно с изменениями в значениях параметров. Другой метод, непосредственное выполнение, используется в тех случаях, когда прикладное обеспечение не запрашивает информацию о результирующем множестве до окончания SQL-запроса, и планируется выполнить из прикладной программы SQL-запрос только один раз.

Выборка результатов – этот набор функций управляет восстановлением данных из результирующего множества SQL-оператора и восстанавливает такую информацию о результирующем множестве как: описание какого-нибудь столбца результирующего множества и его атрибутов, получение следующей строки результирующего множества, подсчет числа строк, на которые воздействует оператор SQL, и т.д. Любая функция использует курсор в извлекаемой таблице или результирующем множестве, указывающий на текущую позицию в строках результирующего множества.

Завершение – функции этого блока предназначены для освобождения памяти, выделенной для идентификаторов при инициализации и завершения соединения с сервером.

Обработка ошибок и сообщения – каждая функция ODBC возвращает в качестве значения возврата признак успешного выполнения, предупреждения или невозможности выполнения.

Возможные коды возврата функций ODBC:

1. **SQL_SUCCESS** – функция выполнена успешно. Информация об ошибке для возврата отсутствует.
2. **SQL_SUCCESS_WITH_INFO** - функция выполнена успешно, однако имеется некоторая информация, которую следует воспринимать как предупреждение.
3. **SQL_NO_DATA_FOUND (SQL_NO_DATA)** – все строки результирующего множества были извлечены. Это не является ошибкой, но указывает, что больше не осталось информации для выборки.
4. **SQL_ERROR** – ошибка в процессе выполнения данной функции. Имеется дополнительная информация об ошибке для восстановления.
5. **SQL_INVALID_HANDLE** – идентификаторы используются в ODBC для представления переменных окружения, соединения и операторов. Эта ошибка возвращается в тех случаях, когда какая-либо функция использует один или более идентификаторов в качестве параметра и один из идентификаторов является недействительным.
6. **SQL_STILL_EXECUTING** – какая-либо функция выполняется асинхронно и все еще находится в процессе выполнения
7. **SQL_INVALID_HANDLE** – ошибка выделения запрошенного идентификатора;
8. **SQL_NEED_DATA** – при подготовке или выполнении какого-либо оператора, драйвер установил, что прикладная программа должна определить не менее одного значения параметра.

Для того, чтобы использовать функции ODBC в своей программе необходимо включить в текст программы заголовочные файлы ODBC API (рис.2.)

```
#include <sql.h>
#include <sqlext.h>
```

Рис.2. Заголовочные файлы для ODBC API

Для получения информации об ошибке используется функция SQLError:

```
RETCODE SQLError(henv,hbdc,hstmt,szSqlState,pfNativeError,szErrorMsg,
                 cbErrorMsgMax,cbErrorMsg)
```

Табл. 1. Параметры SQLError

Тип	Аргумент	Использование	Описание
HENV	henv	Вход	Идентификатор окружения или SQL_NULL_HENV
HDBC	hbdc	Вход	Идентификатор соединения или SQL_NULL_HDBC
HSTMT	hstmt	Вход	Идентификатор оператора или SQL_NULL_HSTMT
UCHAR FAR*	szSqlState	Выход	SQLSTATE в качестве строки завершения
SDWORD FAR*	pfNativeError	Выход	Код ошибки (определяется источником данных)
UCHAR FAR*	szErrorMsg	Выход	Указатель буфера хранения текста сообщения об ошибке
SWORD	cbErrorMsgMax	Вход	Максимальная длина буфера szErrorMsg
SWORD FAR*	cbErrorMsg	Выход	Указатель общего числа байт, которые могут быть использованы при возвращении в szErrorMsg

SQLSTATE – несмотря на то, что текст ошибки дает детальное объяснение о том, какая была ошибка, для программы это не имеет практической пользы. SQLSTATE позволяет прикладной программе идентифицировать обнаруженную ошибку. Иногда это состояние может сообщить программе, как выполнить некоторое действие или исправить ошибку. Состояние является пятибайтной строкой. Каждая из ошибок имеет свой статус. Например, SQLSTATE = “01S00” – неправильная строка с атрибутами соединения, SQLSTATE = “S0022” – столбец уже существует.

Для получения сообщения об ошибке удобно пользовать макросами и функциями, показанными на рис.3.

```
#define RETCODE_IS_FAILURE(x) (x==SQL_ERROR|| \
            x==SQL_INVALID_HANDLE|| \
            x==SQL_STILL_EXECUTING)
#define RETCODE_IS_SUCCESSFULL(x) (!RETCODE_IS_FAILURE(x))

#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include <stdio.h>

void ShowError(HENV hEnv, HDBC hDbc, HSTMT hStmt, UCHAR *pszSqlState)
{
      UCHAR szSqlState[6];
      SDWORD  fNativeError;
      UCHAR szErrorMsg[SQL_MAX_MESSAGE_LENGTH+1];
      SWORD cbErrorMsg;
      RETCODE     rc;
```

```
        char str[SQL_MAX_MESSAGE_LENGTH+1];

        rc=SQL_SUCCESS;
        rc=SQLError(hEnv,hDbc,hStmt,szSqlState,&fNativeError,szErrorMsg,
                        sizeof(szErrorMsg),&cbErrorMsg);
        strcpy((char *)pszSqlState,(char *)szSqlState);
        while(rc!=SQL_NO_DATA_FOUND)
        {
              CharToOem((char *)szErrorMsg,str);
              printf("SQL_ERROR: %s\n",str);
              printf("SqlState SQL_SUCCESS: %s\n",szSqlState);


              rc=SQLError(hEnv,hDbc,hStmt,szSqlState,&fNativeError,szErrorMsg,
                          sizeof(szErrorMsg),&cbErrorMsg);
              strcpy((char *)pszSqlState,(char *)szSqlState);

        }
}
```

Рис.3. Макросы и функция для определения ошибки ODBC

2. Соединение с базой данных

Для соединения с БД необходимо установить в системе драйвер ODBC. Для ведения БД АСКУЭ используется СУБД MySQL и соответственно необходим драйвер ODBC для работы с этой СУБД. Драйвер можно скачать с сайта *www.mysql.com*. В этом примере использовался драйвер ODBC версии 3.51.

Прикладные программы могут устанавливать соединение, как с помощью драйверов, так и с помощью источников данных. В зависимости от типа СУБД и прикладной программы, может оказаться, что драйверное соединение обеспечивает большую функциональность по сравнению с соединением с помощью источника данных. Однако источники данных позволяют более точно определять данное соединение с помощью специфических атрибутов, экономя тем самым время при частых соединениях с БД.

При использовании источника данных необходимо создать источник данных в операционной системе WINDOWS (Панель управления/Администрирование/Источники данных), указав необходимые параметры соединения.

В данном примере будем использовать соединение с помощью драйвера.

2.1. Назначение окружения

Прежде чем, прикладная программа сможет вызвать какую-либо из функций ODBC, она обязана выполнить инициализацию ODBC и установить окружение. ODBC использует это окружение для отслеживания соединения с БД, установленного программой. Поскольку в рамках одного

окружения можно установить произвольное число соединений, для каждой прикладной программы достаточно установить только одно окружение.

Для выделения окружения используется функция SQLAllocHandle:

```
SQLRETURN  SQLAllocHandle(SQLSMALLINT HandleType, SQLHANDLE InputHandle,
                          SQLHANDLE *OutputHandlePtr)
```

Возможные значения параметров SQLAllocHandle:

- ***HandleType –*** задает тип указателя и устанавливается значение SQL_HANDLE_ENV – для выделения идентификатора окружения;
- ***InputHandle*** - входной указатель, в области памяти которого будет выделен идентификатор устанавливается значение SQL_NULL_HANDLE;
- ***OutputHandlePtr*** – выходной указатель на область памяти, в котором будет возвращен выделенный идентификатор.

Возможные коды возврата: SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR (их смысл объяснен в п.1.).

2.2. Назначение идентификатора соединения

Идентификатор соединения представляет соединение между БД и прикладной программой. Для каждой БД, с которой необходимо будет соединиться, необходимо назначать идентификатор соединения.

Для назначения идентификатора соединения используется функция SQLAllocHandle из п.2.1. при этом значения параметров следующие:

- ***HandleType –*** устанавливается значение SQL_HANDLE_DBC – для выделения идентификатора соединения;
- ***InputHandle*** - входной указатель, в области памяти которого будет выделен идентификатор устанавливается значение идентификатора окружения;
- ***OutputHandlePtr*** – выходной указатель на область памяти, в котором будет возвращен идентификатор соединения.

2.3. Соединение с БД

Для соединения с БД будем использовать функцию SQLDriverConnect:

```
SQLRETURN SQLDriverConnect(SQLHDBC ConnectoinHandle,
                           SQLHWND WindowHandle,
                           SQLCHAR *InConnectionString,
                           SQLSMALLINT StringLength1,
                           SQLCHAR * OutConnectionString,
                           SQLSMALLINT BufferLength,
                           SQLSMALLINT *StringLength2Ptr,
                           SQLUSMALLINT fDriverCompletion);
```

Табл. 2. Параметры SQLDriverConnect

Тип	Аргумент	Описание
SQLHDBC	ConnectoinHandle	Идентификатор соединения
SQLHWND	WindowHandle	Идентификатор окна. Идентификатор родительского окна или NULL
SQLCHAR *	InConnectionString	Строка соединения
SQLSMALLINT	StringLength1	Длина строки соединения или SQL_NTS, если строка заканчивается 0
SQLCHAR *	OutConnectionString	Указатель области хранения законченного соединения
SQLUSMALLINT	BufferLength	Длина строки законченного соединения или SQL_NTS, если строка заканчивается 0
SQLSMALLINT *	StringLength2Ptr	Указатель области хранения длины законченного соединения
SQLUSMALLINT	fDriverCompletion	SQL_DRIVER_PROMPT SQL_DRIVER_COMPLETE SQL_DRIVER_COMPLETE_REQUIRED SQL_DRIVER_NOPROMPT

Для соединения с БД функция SQLDriverConnect использует строку соединения (szConStrIn). Строка соединения представляет собой текстовую строку, заканчивающуюся 0, в которой задаются атрибуты соединения, и имеющую следующий формат:

ключевое_слово=значение[;ключевое_слово=значение]

Пример такой строки:

“DRIVER=MySQL ODBC 3.51 Driver;
USER=root;PASSWORD=1;DATABASE=db;
SERVER=192.168.1.2;PORT=3306”,

где

- DRIVER=MySQL ODBC 3.51 Driver – задает имя драйвера ODBC;
- USER=root – определяет имя пользователя для связи с БД;
- PASSWORD=1 - определяет пароль пользователя для связи с БД;
- DATABASE=db – задает имя БД, с которой необходимо связаться;

- SERVER=192.168.1.2;PORT=3306 – определяют соответственно имя или IP-адрес и порт сервера БД.

Параметр fDriverCompletion сообщает менеджеру драйверов ODBC и выбранному драйверу, как завершить строку соединения. Возможные значения этого парамеира:

- SQL_DRIVER_PROMPT – Менеджер драйверов предоставляет окно диалога «Источники данных», для выбора конкретного источника данных для соединения с БД;
- SQL_DRIVER_COMPLETE или SQL_DRIVER_COMPLETE_REQUIRED – если в строке соединения задан параметр DSN (определяет имя источника данных), то Менеджер драйверов использует это имя, иначе вызывает окно диалога, как для первого значения;
- SQL_DRIVER_NOPROMPT - Менеджер драйверов не выполняет приглашения на ввод или ключевое слово DRIVER задано в строке соединения.

```
#include <windows.h>
#include <sql.h>
#include <stdio.h>
#include <sqlext.h>

SQLHENV     henv;
SQLHDBC     hdbc;
SQLRETURN   retcode;
char outcon_str[SQL_MAX_MESSAGE_LENGTH+1];
char incon_str[SQL_MAX_MESSAGE_LENGTH+1];
SQLHWND hwnd=::GetActiveWindow();
SQLSMALLINT len;

int main(int argc, char** argv)
{
      retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
      if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
      {
            if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
            {
                  retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
            }
      }
      if(retcode!=SQL_SUCCESS && retcode!=SQL_SUCCESS_WITH_INFO)
      {
            printf("Ошибка выделения окр. SQL");
            ShowError(NULL,NULL,NULL,szSqlState);
            return 0;
      }

      retcode = SQLDriverConnect(hdbc, hwnd,(SQLCHAR*)incon_str,SQL_NTS,
                  (SQLCHAR*)outcon_str,SQL_NTS,&len,SQL_DRIVER_NOPROMPT);
      if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
      {
            printf("Соединение с БД");
      }
      else
      {
            printf("Ошибка соединения с БД");
```

```
        }

        //Выделение идентификатора оператора,
        //Выполнение SQL-запроса
        //Выборка и анализ данных
        //Освобождение идентификатора оператора

        SQLDisconnect(hdbc);
        SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
        SQLFreeHandle(SQL_HANDLE_ENV, henv);
        return 1;
}
```

Рис.3. Пример программы соединения с БД

Функции SQLFreeHandle и SQLDisconnect будут описаны в следующих подразделах.

2.8. Закрытие соединения с БД

Закрытие соединения с БД осуществляется с помощью функции SQLDisconnect:

```
SQLRETURN  SQLDisconnect(SQLHDBC ConnectionHandle);
```

В качестве значения в параметр ConnectionHandle передается идентификатор уже открытого соединения с БД.

Возможные коды возврата: SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR (их смысл объяснен в п.1.).

2.9. Освобождение идентификаторов окружения и соединения

Для освобождения идентификаторов окружения и соединения используется функция SQLFreeHandle:

```
SQLRETURN  SQLFreeHandle(SQLSMALLINT HandleType, SQLHANDLE Handle);
```

При освобождении идентификатора соединения передаются следующие значения:

- ***HandleType*** - SQL_HANDLE_DBC;
- ***Handle*** – уже выделенный идентификатор соединения.

При освобождении идентификатора окружения передаются следующие значения:

- ***HandleType*** - SQL_HANDLE_ENV;
- ***Handle*** – уже выделенный идентификатор окружения.

Возможные коды возврата: SQL_SUCCESS, SQL_INVALID_HANDLE, SQL_ERROR (их смысл объяснен в п.1.).

3. Чтение и выборка данных из БД

Чтение данных из БД осуществляется с помощью SQL-запросов. Для выполнения SQL-запроса необходимо назначить идентификатор оператора. Любая функция OBDC, которая имеет отношение к обработке или передаче SQL-запросов, требует в качестве параметра идентификатор оператора. ODBC использует два типа функций, связанных с получением данных. Функции первого типа генерируют результаты, а функции второго типа выбирают информацию о результатах и, собственно, результаты.

3.1. Назначение идентификатора оператора

Идентификатор оператора ссылается на SQL-запрос или на другие функции ODBC, которые возвращают результаты.

Для назначения идентификатора оператора используется функция SQLAllocHandle из п.2.1. при этом значения параметров следующие:

- ***HandleType –*** устанавливается значение SQL_HANDLE_STMT – для выделения идентификатора оператора;
- ***InputHandle*** - входной указатель, в области памяти которого будет выделен идентификатор устанавливается значение идентификатора соединения;
- ***OutputHandlePtr*** – выходной указатель на область памяти, в котором будет возвращен идентификатор оператора.

3.2. Создание SQL-запроса

Для чтения данных из БД необходимо выполнить SQL-запрос. В данном примере будет использовано подготавливаемое выполнение (различия между этим типом выполнения и непосредственным описывались в п.1.).

При создании запроса используется язык SQL.

Примечание:

Для создания запроса необходимо знать структуру БД АСКУЭ.

Например,

- для выборки всех данных мгновенных значений по объекту 1 и счетчику со связным номером 1 необходим такой запрос:

```
SELECT * FROM moment_val where n_obj=1 and link_adr=1
```

- для удаления всех данных мгновенных значений по объекту 1 и счетчику со связным номером 1 необходим такой запрос:

```
DELETE * FROM moment_val where n_obj =1 and link_adr =1
```

moment_val – таблица мгновенных значений,

n_obj и link_adr – названия столбцов в таблице для номера объекта и связного номера соответственно;

для выборки всех данных получасовых значений энергии по объекту 1 и счетчику со связным номером 1 за день 12 ноября 2006 г. необходим такой запрос:

```
SELECT * FROM val where n_obj=1 and link_adr=1 and DATA=”2006-11-12”
```

Для подготовки выполнения запроса к БД используется функция SQLPrepare:

```
SQLRETURN  SQLPrepare(SQLHSTMT StatementHandle, SQLCHAR *StatementText,
                      SQLINTEGER TextLength);
```

где ***StatementHandle*** – идентификатор оператора;
StatementText – строка с запросом;
TextLength – длина строки с запросом.

Возможные коды возврата: SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR, SQL_STILL_EXECUTING (их смысл объяснен в п.1.).

3.3. Связывание данных и выполнение запроса

Для получения информации из БД необходимо в прикладной программе определить области хранения для каждой ячейки из записи БД соответствующих типов данных и затем связать эти области с определенными ячейками в БД, указав соответствие драйверу ODBC.

Например, для получения данных из таблицы val (получасовые значения энергии) можно определить следующую структуру:

```
struct  val_30
{
      int   n_obj;            //номер объекта
      int   link_adr;         //связной номер счетчика
      DATE_STRUCT data;       //дата получения значения (день)
      int   interval;         //номер интервала (получаса) за день
      float value;            //значение энергии
      char  priz;             //признак значения
      int   n_zone;           //номер зоны, к которой относится значение
      int   day_type;         //тип дня
};
```

Рис.4. Структура данных для таблицы val

Для связывания данных используется функция SQLBindCol:

```
SQLRETURN  SQLBindCol(SQLHSTMT StatementHandle, SQLUSMALLINT ColumnNumber,
                      SQLSMALLINT TargetType, SQLPOINTER TargetValuePtr,
                      SQLINTEGER BufferLength, SQLLEN *StrLen_or_Ind);
```

Табл. 3. Параметры SQLBindCol

Тип	Аргумент	Описание
SQLHSTMT	StatementHandle	Идентификатор оператора
SQLUSMALLINT	ColumnNumber	Номер столбца в БД
SQLSMALLINT	TargetType	Определяет С-тип данных параметра, не который указывает TargetValuePtr
SQLPOINTER	TargetValuePtr	Указатель на буфер данных, который будет связан с ячейкой БД
SQLINTEGER	BufferLength	Длина буфера данных TargetValuePtr
SQLLEN *	StrLen_or_Ind	Указатель на длину возвращенных данных

Возможные коды возврата: SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR (их смысл объяснен в п.1.).

Пример связывания данных в программе с данными в таблице val:

```
val_30 val_data;
SQLINTEGER cbLen;
SQLHSTMT    hstmt;

SQLBindCol(hstmt,1, SQL_C_SLONG,&val_data.n_obj,0,&cbLen);
SQLBindCol(hstmt,2, SQL_C_SLONG,&val_data.link_adr,0,&cbLen);
SQLBindCol(hstmt,3, SQL_C_TYPE_DATE,&val_data.data,0,&cbLen);
SQLBindCol(hstmt,4, SQL_C_SLONG,&val_data.day_type,0,&cbLen);
SQLBindCol(hstmt,5, SQL_C_SLONG,&val_data.interval,0,&cbLen);
SQLBindCol(hstmt,6, SQL_C_SLONG,&val_data.n_zone,0,&cbLen);
SQLBindCol(hstmt,7, SQL_C_SLONG,&val_data.izm_type,0,&cbLen);
```

```
SQLBindCol(hstmt,8, SQL_C_FLOAT,&val_data.value,0,&cbLen);
SQLBindCol(hstmt,9, SQL_C_CHAR,&val_data.priz,0,&cbLen);
```

Рис.5. Пример связывания данных в программе

Для выполнения подготавливаемого запроса используется функция SQLExecute:

```
SQLRETURN  SQLExecute(SQLHSTMT StatementHandle);
```

где StatementHandle – идентификатор оператора.

Возможные коды возврата: SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR, SQL_STILL_EXECUTING, SQL_NEED_DATA, SQL_NO_DATA (их смысл объяснен в п.1.).

3.4. Выборка данных

После успешного выполнения запроса можно выбирать данные в свою область памяти. Для этой цели служит функция SQLFetch:

```
SQLRETURN  SQLFetch(SQLHSTMT StatementHandle);
```

где ***StatementHandle*** – идентификатор оператора.

Возможные коды возврата: SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR, SQL_STILL_EXECUTING, SQL_NEED_DATA, SQL_NO_DATA (их смысл объяснен в п.1.).

SQLFetch извлекает строку данных из результирующего множества для SQL-запроса. Драйвер возвращает данные для всех столбцов, которые были связаны для предварительного хранения данных с помощью функции SQLBindCol.

На рис. 6 приведена часть программы для выборки всех данных получасовых значений энергии по объекту 1 и счетчику со связным номером 1 за день 12 ноября 2006 г.

```
val_30     val_data;
SQLINTEGER cbLen;
SQLHSTMT   hstmt;
SQLHDBC    hdbc;
SQLRETURN  retcode;
char       str[255];
char       req_energy[]=”SELECT * FROM val where n_obj=1 and link_adr=1\
and DATA=\”2006-11-12\””;

retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
```

```
        retcode=SQLPrepare(hstmt,(SQLCHAR *) req_energy,sizeof(req_energy));
        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
        {
        SQLBindCol(hstmt,1, SQL_C_SLONG,&val_data.n_obj,0,&cbLen);
        SQLBindCol(hstmt,2, SQL_C_SLONG,&val_data.link_adr,0,&cbLen);
        SQLBindCol(hstmt,3, SQL_C_TYPE_DATE,&val_data.data,0,&cbLen);
        SQLBindCol(hstmt,4, SQL_C_SLONG,&val_data.day_type,0,&cbLen);
        SQLBindCol(hstmt,5, SQL_C_SLONG,&val_data.interval,0,&cbLen);
        SQLBindCol(hstmt,6, SQL_C_SLONG,&val_data.n_zone,0,&cbLen);
        SQLBindCol(hstmt,7, SQL_C_SLONG,&val_data.izm_type,0,&cbLen);
        SQLBindCol(hstmt,8, SQL_C_FLOAT,&val_data.value,0,&cbLen);
        SQLBindCol(hstmt,9, SQL_C_CHAR,&val_data.priz,0,&cbLen);
        retcode=SQLExecute(hstmt);
        if (retcode!=SQL_SUCCESS && retcode!=SQL_SUCCESS_WITH_INFO)
        {
              ShowError(henv,hdbc,NULL,szSqlState);
              return 0;
        }
        while(retcode!=SQL_NO_DATA)
        {
              retcode=SQLFetch(hstmt);
              if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
              {
              printf("№об\tСв.№\tТип дня\tТип РИ\t№зоны\t№инт\tЗнач.\tПр\n";
              sprintf(str,"% 7d % 7d % 7d % 7d % 7d % 7d % .2f % c\n",
                      val_data.n_obj, val_data.link_adr, val_data.day_type,
                      val_data.izm_type, val_data.n_zone, val_data.interval,
                      val_data.value, val_data.priz);
              printf(str);
              }
        }
        }
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
}
```

Рис. 6. Часть программы для выборки всех данных получасовых значений энергии

3.5. Освобождение идентификатора оператора

Для освобождения идентификаторов оператора используется функция SQLFreeHandle:

```
SQLRETURN  SQLFreeHandle(SQLSMALLINT HandleType, SQLHANDLE Handle);
```

При освобождении идентификатора соединения передаются следующие значения:

- ***HandleType*** - SQL_HANDLE_STMT;
- ***Handle*** – уже выделенный идентификатор оператора.

Для заметок

Для заметок

ГРАН
СИСТЕМА-С

Республика Беларусь
220141, г. Минск, ул. Ф.Скорины, 54а
Приёмная: тел./факс: (017) 265-82-03
Отдел сбыта: тел. (017) 265-81-87, 265-81-89
Отдел сервиса: тел.: (017) 265 82 09
E-mail: info@strumen.com
http://www.strumen.com

Представительства:
г. Брест, тел. (0162) 42-71-06
г. Витебск, тел. (0212) 24-08-43
г. Гомель, тел. (0232) 48-92-03
г. Гродно, тел. (0152) 79-26-70
г. Могилев, тел. (0222) 28-50-47

v.1.2