ANSI C/ODBC

ODBC Versão 3.51

Fonte:
MSDN

Ambientes de Desenvolvimento Avançados

4º Ano da Licenciatura em Engenharia Informática
Ramos de Sistemas de Informação e de Computadores e Sistemas
Instituto Superior de Engenharia do Porto

Nuno Castro Ferreira

Setembro 2001

ÍNDICE

INTRODUÇÃO

O ODBC (Open Database Connectivity) é uma API (Application Programming Interface) com uma aceitação muito generalizada para acesso a bases de dados. É baseada nas especifícações Call-Level Interface (CLI) do X/Open e ISO/IEC para APIs de acesso a bases de dados. Usa SQL (Structured Query Language) como linguagem de acesso à base de dados.

O ODBC foi desenhado para maximizar a interoperabilidade, isto é, possibilita que uma só aplicação aceda a diversos DBMSs (DataBase Management Systems) usando o mesmo código fonte. As aplicações de base de dados chamam funções do interface do ODBC, que são implementadas em módulos de base de dados específicos chamados drivers. O uso de drivers permite isolar as aplicações de chamadas específicas à base de dados da mesma forma que os drivers de impressoras isolam os programas de processamento de texto das instruções específicas das impressoras. Devido aos drivers só serem carregados em tempo de execução, os utilizadores só necessitam de adicionar um novo driver cada vez que queiram aceder a uma base de dados nova, ou seja, não é necessário recompilar a aplicação.

Esta documentação tem o propósito de fornecer uma ideia base sobre o ODBC, a sua arquitectura e principais funções da API. É aconselhado detalhar os conhecimentos sobre essa API consultando o capítulo do ODBC na MSDN.

ARQUITECTURA ODBC

A arquitectura do ODBC tem quatro componentes:

APLICAÇÃO:

Executa o processamento e chama as funções do ODBC para submeter instruções SQL e obter os resultados.

GESTOR DE DRIVERS:

Carrega e descarrega os drivers a favor de uma aplicação. Processa as funções de ODBC ou passa-as para um driver.

DRIVERS:

Processa as funções do ODBC, submetendo os pedidos de SQL para uma origem de dados específica, retornando os resultados para a aplicação. Se for necessário, o driver modifica o pedido da aplicação de modo a que o pedido esteja em conformidade com a sintaxe suportada pelo DBMS associado.

ORIGEM DOS DADOS:

Consiste nos dados a que os utilizadores pretendem aceder e no sistema operativo que está associado, no DBMS e na plataforma de rede (se existir) usada para aceder ao DBMS.

A figura seguinte mostra a relação entre estes quatro componentes:

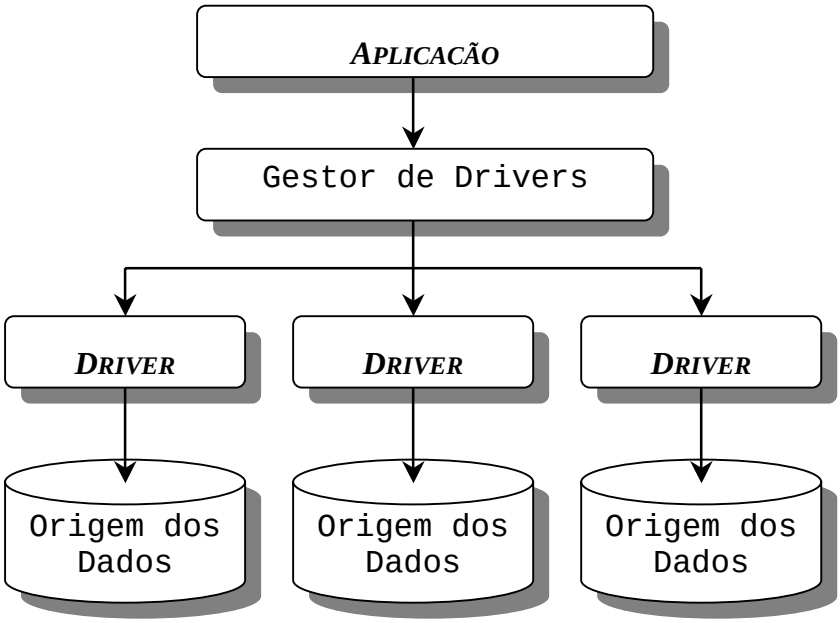
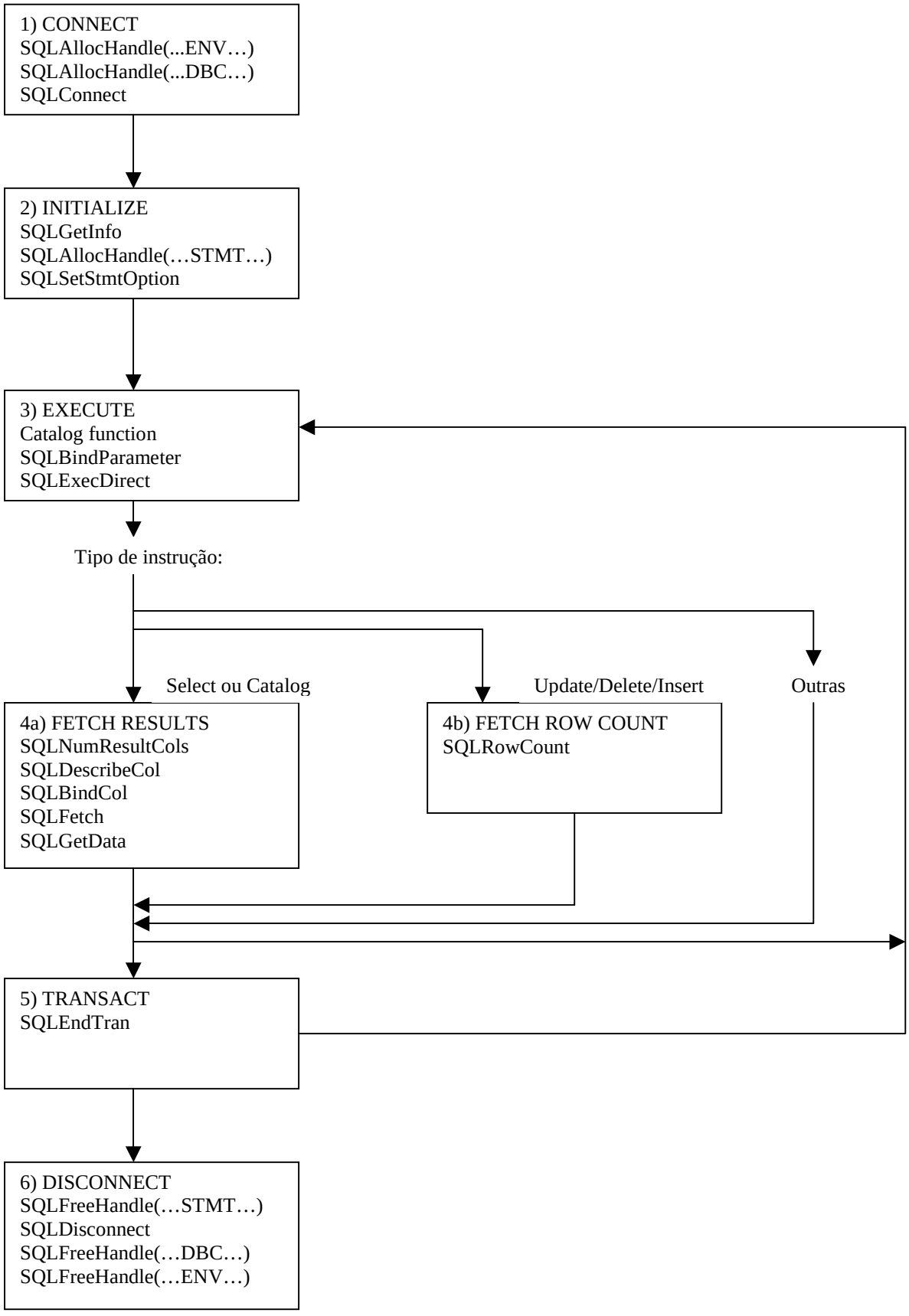


Figura 1 – Relação dos componentes OBDC

Repare no seguinte acerca do diagrama: Primeiro, podem existir múltiplos drivers e origens de dados, o que permite às aplicações acederem simultaneamente a dados provenientes de mais do que uma origem. Em segundo lugar, a API do OBDC é usada em dois lugares: entre a aplicação e o Gestor de Drivers, e entre o Gestor de Drivers e cada driver individual. A ligação entre o Gestor de Drivers e os drivers é por vezes referida como sendo o SPI (*Service Provider Interface*). Para o ODBC, as APIs e os SPIs são os mesmos, ou seja, o Gestor de Drivers e cada driver tem o mesmo interface para as mesmas funções.

ESTRUTURA DE UMA APLICAÇÃO QUE USE OBDC



Primeiro passo:

Inclui a ligação à fonte dos dados (data source).
A função **SQLAllocHandle** permite carregar o driver manager assim como dar inicio à utilização das funções do ODBC. É nessário alocar o Handle de Ambiente (Env) e o de Conexão (DBC).
A função **SQLSetConnectOption** permite inicializar algumas opções globais para a connection, tais como a forma de commit. Se o commit for automático então o ODBC confirma uma transacção automaticamente após a execução de cada statement. Se o commit for manual é necessário chamar a função SQLTransact para fazer commit ou rollback.

Segundo passo:

Neste passo normalmente as aplicações chamam o **SQLGetInfo** para obterem informações sobre as capacidades do driver que estão a utilizar.
Para executar uma instrução SQL é necessário antes efectuar a reserva de um statement através da função **SQLAllocHandle**.
A instrução **SQLSetStmtOption** permite especificar configurações possíveis para o statement, como por exemplo o tipo de cursor ou a forma de gestão de concorrência (locks…).

Terceiro passo:

Normalmente inclui a 'construção' e execução de um comando SQL.
Se o comando SQL tiver parâmetros estes podem ser ligados a variáveis do programa através da função **SQLBindParameter**.
A instrução SQL é executada com a função **SQLExecDirect**.
Se desejarmos que a instrução possa ser executada várias vezes é necessário prepará-la com **SQLPrepare** e só depois executá-la com o **SQLExecute**.

Quarto passo (a):

Se a instrução é do tipo SELECT ou é uma função de Catalog então a função **SQLNumResultCols** permite obter o número de colunas resultantes no result set.
O **SQLDescribeCol** permite obter a descrição de cada coluna resultante: nome, tipo de dado, precisão(número máximo de dígitos) e escala (número máximo de dígitos à direita do ponto decimal).
O **SQLBindCol** permite ligar uma coluna a uma variável do programa.
O **SQLFetch** retorna o valor de um registo para as variáveis do programa indicadas no SQLBindCol.
O **SQLGetData** permite ir buscar valores para os quais não foi efectuado o bind ou valores de colunas com dimensão possivelmente elevada (ex.: campos binários…).

Quarto passo (b):

No caso do comando de SQL for do tipo INSERT, DELETE ou INSERT então como o resultado não é um result set o que se deve fazer é verificar o número de registos afectado com **SQLRowCount**.

Nesta altura a aplicação volta ao passo 3 para executar outra instrução na mesma transacção ou vai para o passo 5 confirmar (commit) ou fazer desfazer (rollback) a transacção.

Quinto passo:

A aplicação confirma (ou desfaz) a transacção com **SQLEndTran**. Só necessita de fazer isso no caso de ter optado por confirmação de transacção manual no primeiro passo.
A aplicação pode então seguir para o passo 3 de forma a executar outro comando SQL ou seguir para o passo 6 e terminar o processamento.

Sexto passo:

Finalmente a aplicação termina o processamento do ODBC.

Primeiro liberta-se o statement com **SQLFreeHandle**.

De seguida 'desliga-se' da data source com **SQLDisconnect**.

Finalmente liberta a connection e o environment com **SQLFreeHandle**.

NÍVEIS DE CONFORMIDADE DO ODBC

CONFORMIDADE DA API

A API do ODBC está dividida em 3 níveis. As funções nucleares incluem todas as funções definidas pela especificação X/Open e SQL Access Group. Para além destas funções o ODBC define ainda dois níveis superiores.

- CORE API

Reservar e libertar handles (connection, environment, statement)
Ligação a fontes de dados. Utilizar vários statements numa ligação.
Preparar e executar instruções SQL. Executar de forma imediata instruções SQL.
Atribuir áreas de armazenamento para parâmetros e colunas de resultado para as instruções SQL.
Obter os dados de um result set. Obter informação sobre um result set.
Commit, rollback sobre transacções.
Obter informação sobre possíveis erros.

- LEVEL 1 API

Toda a funcionalidade da core API.
Ligação a fontes de dados utilizando ecrãs de diálogo específicos do driver.
Obter e alterar valores de opções para statement e connection.
Enviar uma parte ou todo o valor associado a um parâmetro (útil para tipos de dados longos).
Obter parte ou o total de um valor de uma coluna resultante (útil para tipos de dados longos).
Obter informação de catalogo (colunas, colunas especiais, tabelas, etc.).
Obter informação acerca das características e capacidades do driver e da fonte de dados (tipos de dados suportados, funções SQL, funções do ODBC, etc.).

- LEVEL 2 API

Toda a funcionalidade do nível 1 e core.

Poder mostrar a informação da ligação e listar as fontes de dados.

Enviar vectores de valores de parâmetros. Obter vectores de valores de colunas resultantes.

Obter o número de parâmetros e descrever cada parâmetro.

Usar cursores do tipo 'scrollable'.

Obter a forma nativa de uma instrução SQL.

Obter informação de catalogo (privilégios, chaves, procedimentos, etc.).

Chamar uma Dynamic Link Library de tradução.

CONFORMIDADE DO SQL

- MINIMUM SQL GRAMMAR

Data Definition Language (DDL): CREATE TABLE e DROP TABLE.

Data Manipulation Language (DML): SELECT simples, INSERT, UPDATE SEARCHED e DELETE SEARCHED.

Expressões: simples (ex: A > A + C).

Tipos de dados: CHAR, VARCHAR ou LONG VARCHAR.

- CORE SQL GRAMMAR

Gramática e tipos de dados do SQL mínimos.

DDL: ALTER TABLE, CREATE INDEX, DROP INDEX, CREATE VIEW, DROP VIEW, GRANT e REVOKE.

DML: SELECT completo.

Expressões: subquery, funções de conjunto como SUM e MIN.

Tipos de dados: DECIMAL, NUMERIC, SMALLINT, INTEGER, REAL, FLOAT, DOUBLE PRECISION.

- EXTENDED SQL GRAMMAR

Gramática e tipos de dados do SQL mínimo e core.

DML: outer joins, positioned UPDATE, positioned DELETE, SELECT FOR UPDATE e uniões.

Expressões: funções escalares como SUBSTRING E ABS, literais de tempo, datas e timestamp.

Tipos de dados: BIT, TINYINT, BIGINT, BINARY, VARBINARY, LONG VARBINARY, DATE, TIME, TIMESTAMP.

Instruções de SQL em lote.

Chamadas a procedimentos.

- SQLALLOCHANDLE

Descrição:

O **SQLAllocHandle** aloca um handle de environment, connection, statement, ou descriptor.

Sintaxe:

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

Argumentos:

HandleType

[Input]

Tipo de handle a ser alocado por **SQLAllocHandle**. Tem que assumir um dos valores seguintes:

SQL_HANDLE_ENV
SQL_HANDLE_DBC
SQL_HANDLE_STMT
SQL_HANDLE_DESC

InputHandle

[Input]

O handle em cujo contexto o novo handle deve ser alocado. Se *HandleType* é SQL_HANDLE_ENV, este valor será SQL_NULL_HANDLE. Se *HandleType* é SQL_HANDLE_DBC, este valor deverá ser um handle de environment e se é um SQL_HANDLE_STMT ou SQL_HANDLE_DESC, deverá ser um handle de conexão.

OutputHandlePtr

[Output]

Ponteiro para um buffer onde é retornado o handle para a estrutura recém alocada.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, ou SQL_ERROR.

Exemplo:

```
SQLHENV     henv;
SQLHDBC     hdbc;
SQLHSTMT    hstmt;
SQLRETURN   retcode;
...
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
...
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
...
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

- SQLCONNECT

Descrição:

O SQLConnect estabelece uma ligação a um driver e a uma data source.
No handle de conexão está guardada informação sobre a conexão (ex: estado, informação de erros, configuração...).

Sintaxe:

```
SQLRETURN SQLConnect(
      SQLHDBC      ConnectionHandle,
      SQLCHAR *    ServerName,
      SQLSMALLINT NameLength1,
      SQLCHAR *    UserName,
      SQLSMALLINT NameLength2,
      SQLCHAR *    Authentication,
      SQLSMALLINT NameLength3);
```

Argumentos :

ConnectionHandle
[Input]
Handle de conexão.

ServerName
[Input]
Nome da Data source.

NameLength1
[Input]
Tamanho de **ServerName*.

UserName
[Input]
Identificação do utilizador.

NameLength2
[Input]
Tamanho de **UserName*.

Authentication
[Input]
String de autenticação. Normalmente, é a password.

NameLength3
[Input]
Tamanho de **Authentication*.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, ou SQL_INVALID_HANDLE.

Exemplo:

```
retcode = SQLConnect(hdbc,
                  (SQLCHAR*) "Contactos", SQL_NTS,
                  (SQLCHAR*) "GUEST", SQL_NTS,
                  (SQLCHAR*) "ANONYMOUS", SQL_NTS);
```

- SQLDISCONNECT

Descrição:

SQLDisconnect fecha a conexão associada a um handle específico.

Sintaxe:

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

Argumentos:

ConnectionHandle
[Input]
Handle de Conexão.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR ou SQL_INVALID_HANDLE.

Exemplo:

```
SQLDisconnect(hdbc);

SQLFreeHandle(SQL_HANDLE_DBC, hdbc);

SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

- SQLBINDCOL

Descrição:

O **SQLBindCol** associa um buffer de dados a colunas no result set.

Sintaxe:

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

Argumentos :

StatementHandle

[Input]
Handle de statement.

ColumnNumber

[Input]
Número da coluna do result set a associar. As colunas são numeradas ascendentemente a partir de 0, onde a coluna 0 é a coluna de bookmark. Se não forem usados bookmarks, isto é, o atributo SQL_ATTR_USE_BOOKMARKS é definido como SQL_UB_OFF (valor por defeito), então as colunas começam em 1.

TargetType

[Input]
Identificador do tipo de dados em C de **TargetValuePtr*. Nas operações de obtenção de dados com **SQLFetch**, **SQLFetchScroll**, **SQLBulkOperations**, ou **SQLSetPos**, o driver converte os dados para este tipo; nas operações de envio de dados com **SQLBulkOperations** ou **SQLSetPos**, o driver converte os dados a partir deste tipo.

TargetValuePtr

[Deferred Input/Output]
Ponteiro para o buffer de dados a associar a coluna. O **SQLFetch** e o **SQLFetchScroll** retornam os dados neste buffer.

BufferLength

[Input]
Tamanho de **TargetValuePtr* em bytes.

StrLen_or_IndPtr

[Deferred Input/Output]
Ponteiro para o tamanho/indicador do buffer associado à coluna. O SQLFetch e o **SQLFetchScroll** retornam valores neste buffer.

O **SQLFetch** e o **SQLFetchScroll** podem retornam os valores seguintes:

- Tamanho dos dados disponíveis para retornar
- SQL_NO_TOTAL
- SQL_NULL_DATA

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, ou SQL_INVALID_HANDLE.

Exemplo:

```
SQLINTEGER   sCustID, cbCustID;
...
SQLBindCol(hstmt, 1, SQL_C_ULONG, &sCustID, 0, &cbCustID);
```

- SQLPREPARE

Descrição:

O **SQLPrepare** prepara um comando de SQL para execução.

Sintaxe:

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

Argumentos:

StatementHandle
[Input]
Handle de Statement.

StatementText
[Input]
Comando SQL.

TextLength
[Input]
Tamanho de **StatementText*.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, ou SQL_INVALID_HANDLE.

Exemplo:

```
SQLPrepare(hstmt, "{call teste(?)}", SQL_NTS);
```

- SQLEXECUTE

Descrição:

O **SQLExecute** executa um statement já preparado, usando os valores actuais das variáveis dos parameter marker, se existirem.

Sintaxe:

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

Argumentos:

StatementHandle
[Input]
Handle de Statement.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA ou SQL_INVALID_HANDLE.

Exemplo:

```
SQLExecute(hstmt);
```

- SQLEXECDIRECT

Descrição:

O **SQLExecDirect** executa um statement que ainda não foi preparado, usando os valores actuais das variáveis dos parameter marker, se existirem. Engloba o SQLPrepare e o SQLExecute numa só instrução, tornando-se na forma mais rápida de executar commandos SQL de uma só vez.

Sintaxe:

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

Argumentos:

StatementHandle
[Input]
Handle de Statement.
StatementText
[Input]
Comando SQL a ser executado.
TextLength
[Input]
Tamanho de **StatementText*.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA ou SQL_INVALID_HANDLE.

Exemplo:

```
retcode = SQLExecDirect(hstmt,
         "SELECT Numero, Nome, Telefone FROM Clientes ORDER BY 2, 1, 3",
         SQL_NTS);
```

- SQLFETCH

Descrição:

O **SQLFetch** carrega todas as colunas que tenham sigo associadas com os dados do resultado do comando.

Sintaxe:

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

Argumentos:

StatementHandle
[Input]
Handle de Statement.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR ou SQL_INVALID_HANDLE.

Exemplo:

```
retcode = SQLFetch(hstmt);
```

- SQLDESCRIBECOL

Descrição:

O **SQLDescribeCol** retorna a descrição do resultado, ou seja, o nome da coluna, tipo, tamanho, número de casas decimais e suporte a nulos, para uma coluna do result set.

Sintaxe

```
SQLRETURN SQLDescribeCol(
     SQLHSTMT     StatementHandle,
     SQLSMALLINT  ColumnNumber,
     SQLCHAR *    ColumnName,
     SQLSMALLINT   BufferLength,
     SQLSMALLINT * NameLengthPtr,
     SQLSMALLINT * DataTypePtr,
     SQLUINTEGER * ColumnSizePtr,
     SQLSMALLINT * DecimalDigitsPtr,
     SQLSMALLINT * NullablePtr);
```

Argumentos:

StatementHandle
[Input]
Handle statement.

ColumnNumber
[Input]
Número da coluna resultado acerca da qual se pretende obter informação. As colunas estão ordenadas de forma crescente, começando em 1. Se for 0, é obtida a descrição da coluna de bookmark.

ColumnName
[Output]
Ponteiro para um buffer onde é retornado o nome da coluna.

BufferLength
[Input]
Tamanho do buffer **ColumnName* , em caracteres.

NameLengthPtr
[Output]
Ponteiro para um buffer onde é retornado o número total de bytes (excluindo o null terminator) disponíveis para serem retornados em **ColumnName*. Se o número de bytes disponíveis para serem retornados for maior ou igual que *BufferLength*, o nome da coluna em **ColumnName* é truncado para *BufferLength* menos o tamanho do null terminator.

DataTypePtr
[Output]
Ponteiro para um buffer onde é retornado o tipo de dados SQL da coluna.

ColumnSizePtr
[Output]
Ponteiro para um buffer onde é retornado o tamanho da coluna da data source. Se não for possível determinar o tamanho, é retornado 0.

DecimalDigitsPtr
[Output]
Ponteiro para um buffer onde é retornado o número de casas decimais da coluna da data source.. Se este número não for possível de determinar ou não se aplica, é retornado 0.

NullablePtr
[Output]
Ponteiro para um buffer onde é retornada a indicação se a coluna suporta nulos.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Exemplo:

```
char  m_colname[20] = {"\0"};
short m_colsize = 0;
short m_sqltype = 0;
unsigned long m_prec = 0;
short m_scale = 0;
short m_nullable = 0;


retcode = SQLDescribeCol(hstmt, 1,
         (unsigned char *)m_colname, 20,
         &m_colsize, &m_sqltype, &m_prec,
         &m_scale, &m_nullable);
```

- SQLNumResultCols

Descrição:

O **SQLNumResultCols** retorna o número de colunas de um result set.

Sintaxe:

```
SQLRETURN SQLNumResultCols(
     SQLHSTMT     StatementHandle,
     SQLSMALLINT *ColumnCountPtr);
```

Argumentos:

StatementHandle
[Input]
Handle statement.
ColumnCountPtr
[Output]
Ponteiro para um buffer onde é retornado o número de colunas do result set. Esta contagem não inclui a coluna de bookmark.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR ou SQL_INVALID_HANDLE.

Exemplo:

```
retcode = SQLNumResultCols (hstmt, &nResultCols);
```

- SQLFREEHANDLE

Descrição:

O **SQLFreeHandle** liberta os recursos associados com um ambiente, conexão, comando ou descrição específicos.

Sintaxe:

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

Argumentos:

HandleType

[Input]
Tipo de handle a ser libertado. Tem que assumir um dos valores:

SQL_HANDLE_ENV
SQL_HANDLE_DBC
SQL_HANDLE_STMT
SQL_HANDLE_DESC

Se *HandleType* não for um destes valores, **SQLFreeHandle** retorna SQL_INVALID_HANDLE.

Handle

[Input]
Handle a libertar.

Retorno:

SQL_SUCCESS, SQL_ERROR ou SQL_INVALID_HANDLE.

Se **SQLFreeHandle** retornar SQL_ERROR, o handle é ainda válido.

Exemplo:

```
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
```

- SQLGETDIAGREC

Descrição:

O **SQLGetDiagRec** retorna os valores correntes de múltiplos campos de um registo de diagnóstico que contém informação sobre erros, avisos, estados e informações.

Sintaxe:

```
SQLRETURN SQLGetDiagRec(
     SQLSMALLINT   HandleType,
     SQLHANDLE     Handle,
     SQLSMALLINT   RecNumber,
     SQLCHAR *     Sqlstate,
     SQLINTEGER *  NativeErrorPtr,
     SQLCHAR *     MessageText,
     SQLSMALLINT   BufferLength,
     SQLSMALLINT * TextLengthPtr);
```

Argumentos:

HandleType

[Input]

O identificador do tipo de handle que descreve o tipo de handle para o qual é pedido diagnóstico. Tem que ser um dos seguintes:

SQL_HANDLE_ENV
SQL_HANDLE_DBC
SQL_HANDLE_STMT
SQL_HANDLE_DESC

Handle

[Input]

Handle para a estrutura de diagnóstico do tipo indicado por *HandleType*. Se *HandleType* é SQL_HANDLE_ENV, *Handle* pode ser um handle partilhado ou não.

RecNumber

[Input]

Indica o registo de estado acerca do qual é pedido informação. Começa em 1.

SQLState

[Output]

Ponteiro para o buffer onde é retornado o código de 5 caracteres do SQLSTATE. Os dois primeiros caracteres indicam a classe; os outros três indicam a sub-classe.

NativeErrorPtr

[Output]

Ponteiro para o buffer onde é retornado o código de erro nativo, específico para a data source.

MessageText

[Output]

Ponteiro para o buffer onde é retornado o texto de diagnóstico.

BufferLength

[Input]

Tamanho do buffer **MessageText* em caracteres. Não existe um tamanho máximo para a mensagem de diagnóstico.

TextLengthPtr

[Output]

Ponteiro para o buffer onde é retornado o número total bytes (excluindo o número necessário para o null terminator) disponíveis para serem retornados em **MessageText*. Se o número de bytes disponíveis para serem retornados for maior que *BufferLength*, a mensagem de diagnóstico em **MessageText* é truncada para *BufferLength* menos o tamanho do caracter de null terminator.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR ou SQL_INVALID_HANDLE.

Exemplo:

```
SQLCHAR       SqlState[6], Msg[SQL_MAX_MESSAGE_LENGTH];
SQLINTEGER    NativeError;
SQLSMALLINT   i, MsgLen;
SQLRETURN     rc1, rc2;
SQLHSTMT      hstmt;
```

```
if ((rc1 == SQL_SUCCESS_WITH_INFO) || (rc1 == SQL_ERROR)) {
   // Obter o the status records.
   i = 1;
   while ((rc2 = SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, i, SqlState, &NativeError,
            Msg, sizeof(Msg), &MsgLen)) != SQL_NO_DATA)
   {
      DisplayError(SqlState,NativeError,Msg,MsgLen);
      i++;
   }
}
```

- SQLBINDPARAMETER

Descrição:

O **SQLBindParameter** associa um buffer a um parameter marker num comando SQL.

Sintaxe:

```
SQLRETURN SQLBindParameter(
      SQLHSTMT     StatementHandle,
      SQLUSMALLINT ParameterNumber,
      SQLSMALLINT  InputOutputType,
      SQLSMALLINT  ValueType,
      SQLSMALLINT  ParameterType,
      SQLUINTEGER  ColumnSize,
      SQLSMALLINT  DecimalDigits,
      SQLPOINTER   ParameterValuePtr,
      SQLINTEGER   BufferLength,
      SQLINTEGER * StrLen_or_IndPtr);
```

Argumentos:

StatementHandle
[Input]
Handle statement.

ParameterNumber
[Input]
Número do parâmetro, ordenado sequencialmente em ordem crescente, começando em 1.

InputOutputType
[Input]
Tipo do parâmetro. Tem que assumir um dos valores:
SQL_PARAM_INPUT
SQL_PARAM_INPUT_OUTPUT
SQL_PARAM_OUTPUT

ValueType
[Input]
Tipo de dados C do parâmetro.

ParameterType
[Input]
Tipo de dados SQL do parâmetro.
ColumnSize
[Input]
Tamanho da coluna ou expressão correspondente ao parameter marker.
DecimalDigits
[Input]
Número de casas decimais da coluna ou expressão correspondente ao parameter marker.
ParameterValuePtr
[Deferred Input]
Ponteiro para um buffer onde estão os dados para o parâmetro.
BufferLength
[Input/Output]
Tamanho de *ParameterValuePtr* em bytes.
StrLen_or_IndPtr
[Deferred Input]
Ponteiro para um buffer com o tamanho do parâmetro.

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR ou SQL_INVALID_HANDLE.

Exemplo:

```
retcode = SQLBindParameter(hstmt, 1,
                  SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
                  30, 0, szQuote, 0, &cbValue);
```

- SQLRowCount

Descrição:

O **SQLRowCount** retorna o número de registos afectados por um comando de **UPDATE**, **INSERT** ou **DELETE**.

Sintaxe:

```
SQLRETURN SQLRowCount(
     SQLHSTMT      StatementHandle,
     SQLINTEGER * RowCountPtr);
```

Argumentos:

StatementHandle
[Input]
Handle Statement.
RowCountPtr
[Output]
Ponteiro para um buffer onde sera retornado o número de registos

Retorno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR ou SQL_INVALID_HANDLE.

Exemplo:

```
SQLINTEGER RowCount=0;
...
retcode = SQLRowCount(hstmt, &RowCount);
```

- SQLENDTRAN

Descrição:

O **SQLEndTran** efectua um pedido para uma operação de commit ou rollback para todas as operações activas em todos os commandos associados a uma conexão ou a um ambiente.

Sintaxe:

```
SQLRETURN SQLEndTran(
      SQLSMALLINT   HandleType,
      SQLHANDLE     Handle,
      SQLSMALLINT   CompletionType);
```

Argumentos:

HandleType
[Input]
Identificador do tipo de handle. Pode ser SQL_HANDLE_ENV (se *Handle* é de ambiente) ou SQL_HANDLE_DBC (se *Handle* é de conexão).

Handle
[Input]
É o handle, do tipo indicado por *HandleType*, que identifica o limite da transacção.

CompletionType
[Input]
Tem que assumir um dos valores:

SQL_COMMIT
SQL_ROLLBACK

Returno:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR ou SQL_INVALID_HANDLE.

Exemplo:

```
retorno = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
...
retorno = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_ROLLBACK);
```

SQL type identifier	Typical SQL data type	Typical type description
SQL_CHAR	CHAR(n)	Character string of fixed string length n.
SQL_VARCHAR	VARCHAR(n)	Variable-length character string with a maximum string length n.
SQL_LONGVARCHAR	LONG VARCHAR	Variable length character data. Maximum length is data source–dependent.]
SQL_WCHAR	WCHAR(n)	Unicode character string of fixed string length n
SQL_WVARCHAR	VARWCHAR(n)	Unicode variable-length character string with a maximum string length n
SQL_WLONGVARCHAR	LONGWVARCHAR	Unicode variable-length character data. Maximum length is data source–dependent
SQL_DECIMAL	DECIMAL(p,s)	Signed, exact, numeric value with a precision of at least p and scale s. (The maximum precision is driver-defined.) (1 <= p <= 15; s <= p).
SQL_NUMERIC	NUMERIC(p,s)	Signed, exact, numeric value with a precision p and scale s (1 <= p <= 15; s <= p).
SQL_SMALLINT	SMALLINT	Exact numeric value with precision 5 and scale 0 (signed: –32,768 <= n <= 32,767, unsigned: 0 <= n <= 65,535).
SQL_INTEGER	INTEGER	Exact numeric value with precision 10 and scale 0 (signed: –2[31] <= n <= 2[31] – 1, unsigned: 0 <= n <= 2[32] – 1)].
SQL_REAL	REAL	Signed, approximate, numeric value with a binary precision 24 (zero or absolute value 10[–38] to 10[38]).
SQL_FLOAT	FLOAT(p)	Signed, approximate, numeric value with a binary precision of at least p. (The maximum precision is driver-defined.)
SQL_DOUBLE	DOUBLE PRECISION	Signed, approximate, numeric value with a binary precision 53 (zero or absolute value 10[–308] to 10[308]).
SQL_BIT	BIT	Single bit binary data.
SQL_TINYINT	TINYINT	Exact numeric value with precision 3 and scale 0 (signed: –128 <= n <= 127, unsigned:

		0 <= n <= 255).
SQL_BIGINT	BIGINT	Exact numeric value with precision 19 (if signed) or 20 (if unsigned) and scale 0 (signed: –2[63] <= n <= 2[63] – 1, unsigned: 0 <= n <= 2[64] – 1).
SQL_BINARY	BINARY(n)	Binary data of fixed length n.
SQL_VARBINARY	VARBINARY(n)	Variable length binary data of maximum length n. The maximum is set by the user.
SQL_LONGVARBINARY	LONG VARBINARY	Variable length binary data. Maximum length is data source–dependent]
SQL_TYPE_DATE	DATE	Year, month, and day fields, conforming to the rules of the Gregorian calendar
SQL_TYPE_TIME	TIME(p)	Hour, minute, and second fields, with valid values for hours of 00 to 23, valid values for minutes of 00 to 59, and valid values for seconds of 00 to 61. Precision p indicates the seconds precision.
SQL_TYPE_TIMESTAMP[6]	TIMESTAMP(p)	Year, month, day, hour, minute, and second fields, with valid values as defined for the DATE and TIME data types.
SQL_INTERVAL_MONTH	INTERVAL MONTH(p)	Number of months between two dates; p is the interval leading precision.
SQL_INTERVAL_YEAR	INTERVAL YEAR(p)	Number of years between two dates; p is the interval leading precision.
SQL_INTERVAL_YEAR_TO_MONTH	INTERVAL YEAR(p) TO MONTH	Number of years and months between two dates; p is the interval leading precision.
SQL_INTERVAL_DAY	INTERVAL DAY(p)	Number of days between two dates; p is the interval leading precision.
SQL_INTERVAL_HOUR[7]	INTERVAL HOUR(p)	Number of hours between two date/times; p is the interval leading precision.
SQL_INTERVAL_MINUTE[7]	INTERVAL MINUTE(p)	Number of minutes between two date/times; p is the interval leading precision.
SQL_INTERVAL_SECOND	INTERVAL SECOND(p,q)	Number of seconds between two date/times; p is the interval leading precision and q is the interval seconds precision.
SQL_INTERVAL_DAY_TO_HOUR	INTERVAL DAY(p) TO HOUR	Number of days/hours between two date/times; p is the interval leading precision.
SQL_INTERVAL_DAY_TO_MINUTE	INTERVAL DAY(p) TO MINUTE	Number of days/hours/minutes between two date/times; p is the interval leading precision.
SQL_INTERVAL_DAY_TO_SECOND]	INTERVAL DAY(p) TO SECOND(q)	Number of days/hours/minutes/seconds between two date/times; p is the interval leading precision and q is

		the interval seconds precision.
SQL_INTERVAL_HOUR_TO_MINUTE	INTERVAL HOUR(p) TO MINUTE	Number of hours/minutes between two date/times; p is the interval leading precision.
SQL_INTERVAL_HOUR_TO_SECOND	INTERVAL HOUR(p) TO SECOND(q)	Number of hours/minutes/seconds between two date/times; p is the interval leading precision and q is the interval seconds precision.
SQL_INTERVAL_MINUTE_TO_SECOND	INTERVAL MINUTE(p) TO SECOND(q)	Number of minutes/seconds between two date/times; p is the interval leading precision and q is the interval seconds precision.
SQL_GUID	GUID	Fixed length Globally Unique Identifier.

EXEMPLOS

INCLUDES NECESSÁRIOS:

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

MINI - PROGRAMA COMPLETO:

```
void main()
{
	SQLRETURN	rcSQL=0;
	SQLHDBC		hDBC=0;
	SQLHENV		hEnv=0;
	SQLHSTMT	hStmt=0;

	SQLINTEGER TamNr=0;
	SQLINTEGER TamNome=0;

	double NrAluno=0;
	char NomeAluno[80];

	char Servidor[10];

	strcpy(Servidor, "ISEP");

	/* Alocar o Ambiente */
	rcSQL=SQLAllocEnv(&hEnv);
	if (rcSQL != SQL_SUCCESS )
	{

	}

	/* Alocar a conexão */
	rcSQL=SQLAllocConnect (hEnv, &hDBC);
	if (rcSQL != SQL_SUCCESS )
	{

	}

	/* Ligar */
	rcSQL = SQLConnect (hDBC, (unsigned char*)Servidor, SQL_NTS, NULL, SQL_NTS,
NULL, SQL_NTS);
	if (rcSQL < SQL_SUCCESS )
	{

	}

	/* Alocar o comando */
	rcSQL = SQLAllocStmt (hDBC, &hStmt);
	if (rcSQL != SQL_SUCCESS)
	{
		_Ve_Erro(0, hDBC, hEnv, rcSQL);
	}

	/* Executar */
	rcSQL=SQLExecDirect(hStmt,
			(unsigned char*) “SELECT NR, NOME FROM ALUNOS”,
			SQL_NTS);
```

```
        if (rcSQL != SQL_SUCCESS)
        {
                _Ve_Erro(hStmt, hDBC, hEnv, rcSQL);
        }

        /* Associar as colunas */
        TamNr=sizeof(double);
        rcSQL = SQLBindCol(hStmt,
                        1,
                        SQL_C_DOUBLE,
                        &NrAluno,
                        sizeof(NrAluno),
                        &TamNr);

        TamNome=sizeof(NomeAluno) +1 ; /* Por causa do \n */
        rcSQL = SQLBindCol(hStmt,
                        2,
                        SQL_C_CHAR,
                        NomeAluno,
                        TamNome,
                        &TamNome);

        do
        {
                /* Recuperar os dados */
                rcSQL=SQLFetch(hStmt);
                if ( (rcSQL == SQL_SUCCESS) && (rcSQL != SQL_NO_DATA_FOUND))
                {
                        /* Processar */

                }
        }
        while( (rcSQL == SQL_SUCCESS) && (rcSQL != SQL_NO_DATA_FOUND) );

        /* Fechar o comando */
        SQLFreeStmt(hStmt, SQL_CLOSE);
        SQLFreeHandle(SQL_HANDLE_STMT, hStmt);


        /* Desligar */
        rcSQL = SQLDisconnect(hDBC);
        if (rcSQL != SQL_SUCCESS )
        {

        }

        /* Libertar os Handles */
        rcSQL = SQLFreeHandle(SQL_HANDLE_DBC , hDBC);
        if (rcSQL != SQL_SUCCESS )
        {

        }

        rcSQL = SQLFreeHandle(SQL_HANDLE_ENV , hEnv);
        if (rcSQL != SQL_SUCCESS )
        {

        }
}


/* Vizualizar os erros */
void _Ve_Erro(SQLHSTMT hStmt, SQLHDBC hDBC, SQLHENV hEnv, SQLRETURN rc)
{

        SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1];
        SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1];
        SQLINTEGER  sqlcode;
        SQLSMALLINT length;

        SQLSMALLINT i = 1;
        SQLRETURN rc2;

        if (rc == SQL_SUCCESS_WITH_INFO)
        {
                puts("SQL_SUCCESS_WITH_INFO");
        }
```

```
        if (rc == SQL_NO_DATA)
        {
                puts("SQL_NO_DATA");
        }
        if (rc == SQL_ERROR)
        {
                puts("SQL_ERROR");
        }
        if (rc == SQL_INVALID_HANDLE)
        {
                puts("SQL_INVALID_HANDLE");
        }

        while ((rc2 = SQLGetDiagRec(SQL_HANDLE_STMT,
                        hStmt,
                        i,
                        sqlstate,
                        &sqlcode,
                        buffer,
                        sizeof(buffer),
                        &length)) != SQL_NO_DATA)
        {
                puts((const char*)buffer);
                i++;
        }

        return;
}
```