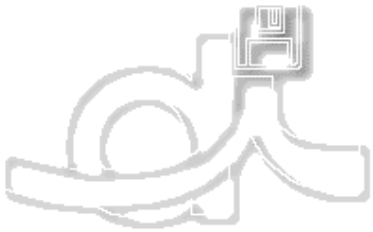

Programação com ODBC 3a

Ambientes de Desenvolvimento Avançados

Engenharia Informática
Instituto Superior de Engenharia do Porto

Alexandre Bragança
1998/99

Drivers de ODBC existentes numa máquina

```
HKEY_LOCAL_MACHINE
     SOFTWARE
          ODBC
              Odbcinst.ini
               Text : REG_SZ : Installed
                     APILevel : REG_SZ : 1
                     ConnectFunctions : REG_SZ : YYN
                     Driver : REG_SZ : C:\WINDOWS\SYSTEM32\TEXT.DLL
                     DriverODBCVer : REG_SZ : 03.00.00
                     FileExtns : REG_SZ : *.txt,*.csv
                     FileUsage : REG_SZ : 1
                     Setup : REG_SZ :
                     C:\WINDOWS\SYSTEM32\TXTSETUP.DLL
                     SQLLevel : REG_SZ : 0
                     UsageCount : REG_DWORD : 0x3
               SQL Server : REG_SZ : Installed
```

DSN de ODBC configurados numa máquina

```
HKEY_LOCAL_MACHINE
     SOFTWARE
          ODBC
               Odbc.ini
Inventory : REG_SZ : SQL Server
      Description : REG_SZ : Inventory database on server InvServ
      Driver : REG_SZ : C:\WINDOWS\SYSTEM32\SQLSRV32.DLL
      OEMTOANSI : REG_SZ : Yes
      Server : REG_SZ : InvServ
      TranslationDLL : REG_SZ : C:\WINDOWS\SYSTEM32\MSCPXL32.DLL
      TranslationName : REG_SZ : MS Code Page Translator
      TranslationOption : REG_SZ : 12500850

Payroll : REG_SZ : dBASE
Personnel : REG_SZ : Text


   HKEY_CURRENT_USER
        SOFTWARE
             ODBC
                  Odbc.ini
```

API do ODBC

SQLConnect

Permite estabelecer a ligação a um Driver e DSN (Data Source Name).

Apenas permite especificar 3 parametros: DSN, Utilizador e Password.

Exemplo:

```
SQLHENV     henv;  SQLHDBC     hdbc;  SQLHSTMT    hstmt;
SQLRETURN   retcode;

/*Allocate environment handle */
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
   /* Set the ODBC version environment attribute */
   retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                     (void*)SQL_OV_ODBC3, 0);

   if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
   {
      /* Allocate connection handle */
      retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

      if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
      {
         /* Set login timeout to 5 seconds. */
         SQLSetConnectAttr(hdbc, (void*)SQL_LOGIN_TIMEOUT, 5, 0);

         /* Connect to data source */
         retcode = SQLConnect(hdbc, (SQLCHAR*) "Sales", SQL_NTS,
                   (SQLCHAR*) "JohnS", SQL_NTS,
                   (SQLCHAR*) "Sesame", SQL_NTS);

         if (retcode == SQL_SUCCESS || retcode ==
                SQL_SUCCESS_WITH_INFO)
         {
            /* Allocate statement handle */
            retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

            if (retcode == SQL_SUCCESS || retcode ==
               SQL_SUCCESS_WITH_INFO)
            {
               /* Process data */
               ;
               SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
            }
            SQLDisconnect(hdbc);
         }
         SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
      }
   }
   SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

SQLDriverConnect

Permite estabelecer uma ligação a um Data Source que necessite de mais dados que os possiveis de passar pelo SQLConnect.

Permite ainda abrir ligações que estão especificadas em ficheiros .DSN (e tambem gerar ficheiros .DSN).

Permite estabelecer ligações a data sources apenas pelo nome do Driver, não utilizando DSN.

SQLBrowseConnect

Permite que interactivamente o programa vá especificando todos os dados necessários para estabelecer uma ligação.

SQLDrivers

Devolve todos os drivers existentes no sistema (1 de cada vez).

SQLDataSources

Devolve informação sobre as DSN existentes no sistema (1 de cada vez)

SQLGetInfo

Permite obter informação sobre o driver e o DSN.

Exemplo:
SQL_ODBC_INTERFACE_CONFORMANCE
SQL_SERVER_NAME
SQL_SQL_CONFORMANCE

SQLGetTypeInfo

Permite saber os tipos de dados suportados pelo DSN.
O Resultado é devolvido na forma de um result set.

SQLPrepare

Permite preparar uma instrução sql para execução futura. Uma instrução preparada pode ser executada várias vezes e pode conter parametros.

Deve-se passar o texto da instrução. No caso de termos parametros estes são assinalados atraves de *parameter markers* (?)

Exemplo:

```
#define SALES_PERSON_LEN 10
#define STATUS_LEN 6

SQLSMALLINT      sOrderID;
SQLSMALLINT      sCustID;
DATE_STRUCT      dsOpenDate;
SQLCHAR         szSalesPerson[SALES_PERSON_LEN];
SQLCHAR         szStatus[STATUS_LEN);
SQLINTEGER      cbOrderID = 0, cbCustID = 0, cbOpenDate = 0,
cbSalesPerson = SQL_NTS,
cbStatus = SQL_NTS;
SQLRETURN retcode;
SQLHSTMT hstmt;

/* Prepare the SQL statement with parameter markers. */
retcode = SQLPrepare(hstmt,
        "INSERT INTO ORDERS (ORDERID, CUSTID, OPENDATE, SALESPERSON,
        STATUS) VALUES (?, ?, ?, ?, ?)", SQL_NTS);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
        /* Specify data types and buffers for OrderID, CustID,
        OpenDate, SalesPerson, */
        /* Status parameter data. */
        SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_SSHORT,
                SQL_INTEGER, 0, 0, &sOrderID, 0, &cbOrderID);
        SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_SSHORT,
                SQL_INTEGER, 0, 0, &sCustID, 0, &cbCustID);
        SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_TYPE_DATE,
                SQL_TYPE_DATE, 0, 0, &dsOpenDate, 0, &cbOpenDate);
        SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
                SQL_CHAR, SALES_PERSON_LEN, 0, szSalesPerson, 0,
                &cbSalesPerson);
        SQLBindParameter(hstmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR,
                SQL_CHAR, STATUS_LEN, 0, szStatus, 0, &cbStatus);

        /* Specify first row of parameter data. */
        sOrderID = 1001;
        sCustID = 298;
        dsOpenDate.year = 1996;
        dsOpenDate.month = 3;
        dsOpenDate.day = 8;
        strcpy(szSalesPerson, "Johnson");
        strcpy(szStatus, "Closed");



        /* Execute statement with first row. */
        retcode = SQLExecute(hstmt);
```

```
        /* Specify second row of parameter data. */
        sOrderID = 1002;
        sCustID = 501;
        dsOpenDate.year = 1996;
        dsOpenDate.month = 3;
        dsOpenDate.day = 9;
        strcpy(szSalesPerson, "Bailey");
        strcpy(szStatus, "Open");

        /* Execute statement with second row. */
        retcode = SQLExecute(hstmt);
}
```

SQLBindParameter

Permite ligar um buffer (variavel) de um programa a um parameter marker de uma instrução.

Deve-se especificar se o parametro e de INPUT, INPUT/OUTPUT ou OUTPUT.
Deve-se ainda especificar o tipo de dado da variavel do programa e o tipo de dado do parametro.

SQLExecute

Permite executar uma instrução preparada utilizando os valores actuais dos parameter markers.

SQLExecDirect

Utilizado para executar directamente instruções (sem preparação anterior).

SQLDescribeParam

Permite obter descrição sobre os parametros de uma instrução, nomeadamente o tipo de dado.

SQLNumParams

Retorna o numero de parametros numa instrução.

SQLRowCount

Permite saber o numero de registos afectado por instruções do tipo insert, update ou delete.

SQLNumResultCols

Permite saber o numero de colunas de um result set.

SQLDescribeCol

Permite obter informação sobre as colunas de um result set, nomeadamente o tipo de dado.

SQLFetch

Permite 'ir buscar' o proximo registo de dados de um result set e actualizar os dados de todas as colunas para as quais se invocou o SQLBindCol.

SQLBindCol

Permite ligar um buffer (variavel) de um programa a uma coluna de um result set.

Exemplo:

```
#define NAME_LEN 50
#define PHONE_LEN 10

SQLCHAR      szName[NAME_LEN], szPhone[PHONE_LEN];
SQLINTEGER   sCustID, cbName, cbCustID, cbPhone;
SQLHSTMT      hstmt;
SQLRETURN   retcode;

retcode = SQLExecDirect(hstmt,
            "SELECT CUSTID, NAME, PHONE FROM CUSTOMERS ORDER BY 2, 1,
               3", SQL_NTS);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
        /* Bind columns 1, 2, and 3 */
        SQLBindCol(hstmt, 1, SQL_C_ULONG, &sCustID, 0, &cbCustID);
        SQLBindCol(hstmt, 2, SQL_C_CHAR, szName, NAME_LEN, &cbName);
        SQLBindCol(hstmt, 3, SQL_C_CHAR, szPhone, PHONE_LEN,
                &cbPhone);

        /* Fetch and print each row of data. On */
        /* an error, display a message and exit. */
        while (TRUE)
        {
                retcode = SQLFetch(hstmt);
                if (retcode == SQL_ERROR || retcode ==
                        SQL_SUCCESS_WITH_INFO)
                {
                        show_error();
                 }

                if (retcode == SQL_SUCCESS || retcode ==
                        SQL_SUCCESS_WITH_INFO)
                {
                        fprintf(out, "%-*s %-5d %*s", NAME_LEN-1,
                                szName, sCustID, PHONE_LEN-1,
                                szPhone);
                }
                else
                {
                        break;
                }
        }
 }
```

SQLFetchScroll

Permite 'ir buscar' um conjunto de registos (rowset) de um result set para todas as colunas para as quais se tenha invocado o SQLBindCol.

Ao contrário do Fetch o FetchScroll permite vários 'movimentos':

SQL_FETCH_NEXT
SQL_FETCH_PRIOR
SQL_FETCH_FIRST
SQL_FETCH_LAST
SQL_FETCH_ABSOLUTE
SQL_FETCH_RELATIVE
SQL_FETCH_BOOKMARK

Exemplo:

```
#define UPDATE_ROW   100
#define DELETE_ROW   101
#define ADD_ROW      102

SQLUINTEGER    CustIDArray[11];
SQLCHAR        NameArray[11][51], AddressArray[11][51],
               PhoneArray[11][11];
SQLINTEGER     CustIDIndArray[11], NameLenOrIndArray[11],
               AddressLenOrIndArray[11],
               PhoneLenOrIndArray[11];
SQLUSMALLINT   RowStatusArray[10], Action, RowNum;
SQLRETURN      rc;
SQLHSTMT       hstmt;

// Set the SQL_ATTR_ROW_BIND_TYPE statement attribute to use column-
// wise
// binding. Declare the rowset size with the SQL_ATTR_ROW_ARRAY_SIZE
// statement attribute. Set the SQL_ATTR_ROW_STATUS_PTR statement
// attribute to point to the row status array.

SQLSetStmtAttr(hstmt, SQL_ATTR_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN,
                      0);
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_BIND_TYPE, SQL_BIND_BY_COLUMN, 0);
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE, 10, 0);
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_STATUS_PTR, RowStatusArray, 0);

// Bind arrays to the CustID, Name, Address, and Phone columns.
SQLBindCol(hstmt, 1, SQL_C_ULONG, CustIDArray, 0, CustIDIndArray);
SQLBindCol(hstmt, 2, SQL_C_CHAR, NameArray, sizeof(NameArray[0]),
                NameLenOrIndArray);
SQLBindCol(hstmt, 3, SQL_C_CHAR, AddressArray,
                sizeof(AddressArray[0]),
               AddressLenOrIndArray);
SQLBindCol(hstmt, 4, SQL_C_CHAR, PhoneArray, sizeof(PhoneArray[0]),
           PhoneLenOrIndArray);

// Execute a statement to retrieve rows from the Customers table.
SQLExecDirect(hstmt, "SELECT CustID, Name, Address, Phone FROM
                      Customers", SQL_NTS);



// Fetch and display the first 10 rows.
```

```
rc = SQLFetchScroll(hstmt, SQL_FETCH_NEXT, 0);

DisplayData(CustIDArray, CustIDIndArray, NameArray,
               NameLenOrIndArray, AddressArray,
               AddressLenOrIndArray, PhoneArray, PhoneLenOrIndArray,
               RowStatusArray);

// Call GetAction to get an action and a row number from the user.
while (GetAction(&Action, &RowNum)) {
   switch (Action) {

      case SQL_FETCH_NEXT:
      case SQL_FETCH_PRIOR:
      case SQL_FETCH_FIRST:
      case SQL_FETCH_LAST:
      case SQL_FETCH_ABSOLUTE:
      case SQL_FETCH_RELATIVE:
         // Fetch and display the requested data.
         SQLFetchScroll(hstmt, Action, RowNum);
         DisplayData(CustIDArray, CustIDIndArray,
                     NameArray, NameLenOrIndArray,
                     AddressArray, AddressLenOrIndArray,
                     PhoneArray, PhoneLenOrIndArray, RowStatusArray);
         break;

      case UPDATE_ROW:
         // Place the new data in the rowset buffers and update the
         // specified row.
         GetNewData(&CustIDArray[RowNum - 1],
                      &CustIDIndArray[RowNum - 1],
                       NameArray[RowNum - 1],
                      &NameLenOrIndArray[RowNum - 1],
                       AddressArray[RowNum - 1],
                      &AddressLenOrIndArray[RowNum - 1],
                      PhoneArray[RowNum - 1],
                      &PhoneLenOrIndArray[RowNum - 1]);

         SQLSetPos(hstmt, RowNum, SQL_UPDATE, SQL_LOCK_NO_CHANGE);
         break;

      case DELETE_ROW:
         // Delete the specified row.
         SQLSetPos(hstmt, RowNum, SQL_DELETE, SQL_LOCK_NO_CHANGE);
         break;

      case ADD_ROW:
         // Place the new data in the rowset buffers at index 10.
         // This is an extra element for new rows so rowset data is
         // not overwritten. Insert the new row. Row 11 corresponds
         // to index 10.
         GetNewData(&CustIDArray[10], &CustIDIndArray[10],
                     NameArray[10], &NameLenOrIndArray[10],
                     AddressArray[10], &AddressLenOrIndArray[10],
                     PhoneArray[10], &PhoneLenOrIndArray[10]);
         SQLSetPos(hstmt, 11, SQL_ADD, SQL_LOCK_NO_CHANGE);
         break;
   }
}

// Close the cursor.
SQLCloseCursor(hstmt);
```

SQLSetPos

Permite alterar a posição do cursor dentro de um rowset e refrescar, actualizar ou apagar os dados de um result set.

As operações possiveis:
SQL_POSITION
SQL_REFRESH
SQL_UPDATE
SQL_DELETE

Tipo de lock do registo alterado apos a operação:
SQL_LOCK_NO_CHANGE
SQL_LOCK_EXCLUSIVE
SQL_LOCK_UNLOCK

Exemplo:

```
#define ROWS 20
#define STATUS_LEN 6

SQLCHAR        szStatus[ROWS][STATUS_LEN], szReply[3];
SQLINTEGER     cbStatus[ROWS], cbOrderID;
SQLUSMALLINT   rgfRowStatus[ROWS];
SQLUINTEGER    sOrderID, crow = ROWS, irow;
SQLHSTMT       hstmtS, hstmtU;

SQLSetStmtAttr(hstmtS, SQL_ATTR_CONCURRENCY, (SQLPOINTER)
               SQL_CONCUR_ROWVER, 0);
SQLSetStmtAttr(hstmtS, SQL_ATTR_CURSOR_TYPE, (SQLPOINTER)
               SQL_CURSOR_KEYSET_DRIVEN, 0);
SQLSetStmtAttr(hstmtS, SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER) ROWS,
               0);
SQLSetStmtAttr(hstmtS, SQL_ATTR_ROW_STATUS_PTR, (SQLPOINTER)
               rgfRowStatus, 0);

SQLSetCursorName(hstmtS, "C1", SQL_NTS);

SQLExecDirect(hstmtS, "SELECT ORDERID, STATUS FROM ORDERS ",
               SQL_NTS);

SQLBindCol(hstmtS, 1, SQL_C_ULONG, &sOrderID, 0, &cbOrderID);
SQLBindCol(hstmtS, 2, SQL_C_CHAR, szStatus, STATUS_LEN, &cbStatus);

while ((retcode == SQLFetchScroll(hstmtS, SQL_FETCH_NEXT, 0)) !=
        SQL_ERROR)
{
   if (retcode == SQL_NO_DATA_FOUND)
      break;
   for (irow = 0; irow < crow; irow++)
   {
      if (rgfRowStatus[irow] != SQL_ROW_DELETED)
         printf("%2d %5d %*s\n", irow+1, sOrderID, NAME_LEN-1,
               szStatus[irow]);
   }



   while (TRUE)
```

```
   {
      printf("\nRow number to update?");
      gets(szReply);
      irow = atoi(szReply);
      if (irow > 0 && irow <= crow)
      {
         printf("\nNew status?");
         gets(szStatus[irow-1]);
         SQLSetPos(hstmtS, irow, SQL_POSITION, SQL_LOCK_NO_CHANGE);
         SQLPrepare(hstmtU,
                  "UPDATE ORDERS SET STATUS=? WHERE CURRENT OF C1",
                   SQL_NTS);
         SQLBindParameter(hstmtU, 1, SQL_PARAM_INPUT,
            SQL_C_CHAR, SQL_CHAR,
            STATUS_LEN, 0, szStatus[irow], 0, NULL);
         SQLExecute(hstmtU);
      }
      else if (irow == 0)
      {
         break;
      }
   }
}
```

SQLGetDiagRec

Permite obter informação de diagnostico ou erro.

SQLEndTran

Permite efectuar um commit ou rollback sobre todas as operações activas de todas as instruções associadas com uma conexão.