Queries with Dynamic Parameters

ODBC applications frequently require data to be sent to the database dynamically. Such data cannot be hard-coded in the query itself.

For comparison, the following set of queries inserts three rows with hard-coded parameters:

```
insert into cars( model ) values( 'Hummer' );
insert into cars( model ) values( 'Mustang' );
insert into cars( model ) values( 'Camray' );
```

The above set of queries would be adequate if it was known in advance that the target table would always receive this data. Often this is not the case, so the data must be sent dynamically. The ODBC uses question marks ("?") as placeholders for dynamic queries:

```
insert into cars( model ) values ( ? );
```

A character string can be defined and *bound* to the query. Binding a parameter to a query means that the data is sent as a parameter along with the query. The DBMS reconstructs the parameter data with the query as if the data were hard-coded in the original query.

To send data to the DBMS, the ODBC must tell the DBMS what the data looks like. The ODBC function SQLBindParameter() serves that purpose. The following example allows an ODBC application to send one row of data dynamically to a table consisting of an integer and a varchar of length 20:

```
SQLCHAR model[21] = "Mustang ";
int lotNbr = 124;
SQLINTEGER orind1 = 0, orind2 = SQL_NTS;

[ Allocate handles and connect. ]

SQLExecDirect( hstmt, "insert into cars( model ) values( ? )", SQL_NTS );

SQLBindParameter( hstmt, /* Statement handle */
    1,                   /* Column number 1 */
    SQL_PARAM_INPUT,     /* This is an input parameter */
    SQL_C_LONG,          /* This is an integer in C */
    SQL_INTEGER,         /* Destination column is varchar */
    strlen( model ),     /* Length of the parameter */
    0,                   /* No scale specifier */
    model,               /* The data itself */
    0,                   /* Maximum length (default 0) */
    &orind1 );           /* Null-terminated string */

SQLBindParameter( hstmt, /* Statement handle */
    2,                   /* Column number 1 */
    SQL_PARAM_INPUT,     /* This is an input parameter */
    SQL_C_CHAR,          /* This is a string in C */
    SQL_VARCHAR,         /* Destination column is varchar */
    strlen( model[i] ),  /* Length of the parameter */
    0,                   /* No scale specifier */
    model,               /* The data itself */
    0,                   /* Maximum length (default 0) */
    &orind2 );           /* Null-terminated string */
```

Dynamic parameters can be used in WHERE clauses. The following example selects from the cars table, using a dynamic parameter. This query is known as a searched query:

```
SQLCHAR model[21] = "Hummer";
int i;
SQLINTEGER orind = SQL_NTS;

[ Allocate handles and connect. ]

SQLExecDirect( hstmt, "select model from cars where model =
    ( ? )", SQL_NTS );

SQLBindParameter( hstmt,  /* Statement handle */
    1,                    /* Column number 1 */
    SQL_PARAM_INPUT,      /* This is an input parameter */
    SQL_C_CHAR,           /* This is a string in C */
    SQL_VARCHAR,          /* Destination column is varchar */
    strlen( model ),      /* Length of the parameter */
    0,                    /* No precision specifier */
    model,                /* The data itself */
    0,                    /* Maximum length (default 0) */
    &orind );             /* Null-terminated string */
```

Faster inserts also can be achieved if the following conditions are met:

- Inserts must be into a base table (not a view or index).
- The table must not have any rules or integrities defined on it.
- The table must not be a gateway table (for example, an IMA table, security audit log file, or an Enterprise Access table).