# TestPoint™ Database Toolkit

# User's Manual

Limited Warranty

Capital Equipment Corporation (CEC) warrants the physical diskettes and documentation enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date. The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective diskettes or documentation, and shall not include or extend to any claim for or right to recover any other damagaes, including but no limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims. In no event shall CEC's liability exceed the purchase price of the product.



TestPoint Database Toolkit
Second edition

Capital Equipment Corporation
900 Middlesex Turnpike, Building 2
Billerica, Massachusetts 01821
(978) 663-2002

Chapter 1. Installation ..... 1-1
Enabling the Toolkit ..... 1-3
Optional - Add the Database object to your Stock window ..... 1-5
Chapter 2. Introduction ..... 2-1
Chapter 3. Setting up ODBC Data Sources ..... 3-1
Chapter 4. Example - Querying a Database ..... 4-1
Chapter 5. Tutorial - Databases and SQL ..... 5-1
Databases, Standards, and TestPoint ..... 5-2
Database Tutorial ..... 5-3
SQL - Structured Query Language ..... 5-6
Letting the Database program write SQL statements for you ..... 5-12
Chapter 6. TestPoint Database Utilities ..... 6-1
Viewing Data Source Names ..... 6-2
Viewing Tables in a Database ..... 6-3
Viewing Column (Field) Information ..... 6-4
Chapter 7. The Database Object ..... 7-1
Settings ..... 7-2
Actions ..... 7-4
Chapter 8. Examples ..... 8-1
Query and Display Results ..... 8-2
Query and Retrieve Single Values in a Loop ..... 8-4
Using Two or More Database Objects at Once ..... 8-5
Chapter 9. Using Parameters to Pass Data into the Database ..... 9-1

SQL Parameter Syntax....9-2
Using SQL Parameters in TestPoint....9-3
TestPoint and SQL Data Types....9-4

Chapter 10. More Examples....10-1
Inserting Records Into a Database....10-2
Queries with Variable Parameters....10-3

Chapter 11. Notes on Specific Database Programs....11-1
Microsoft Access....11-2
Microsoft SQL Server....11-3
Oracle....11-4

Chapter 12. Distributing Runtimes....12-1
Licensing Policy....12-2
Creating a runtime - Files to distribute....12-3
Registering the database servers....12-4

Chapter 13. Troubleshooting....13-1
Error Messages....13-2

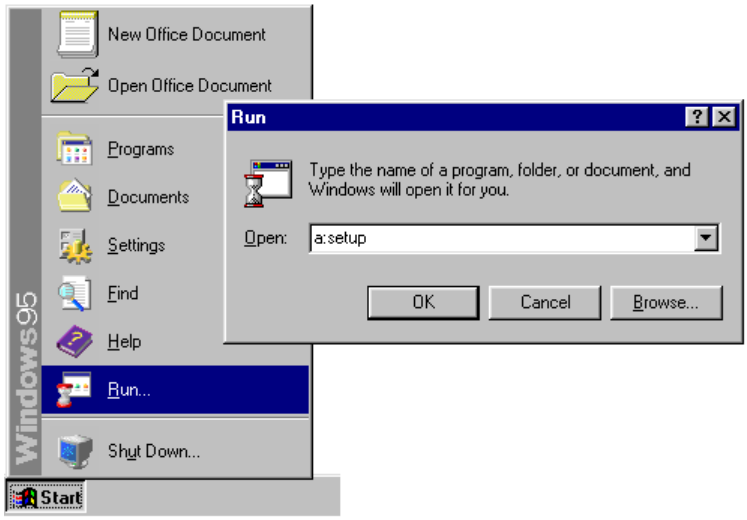
# *Chapter 1. Installation*

## System Requirements

- TestPoint version 3.0 or later
- Windows 3.x, 95, 98, or NT
- Any ODBC-compatible database program, with ODBC drivers
- At least 8M RAM, and 1M free disk space

## Setup

To install the TestPoint Database Toolkit, run SETUP from the distribution diskette. In Windows 3.x, you use the File/Run menu command in Program Manager. In Windows 95, 98, or NT, you use the Start button and Run command as shown below:




Install the Database Toolkit into the **same** directory as your existing TestPoint development system.

## *Enabling the Toolkit*

If you are installing this toolkit as an addition to an existing TestPoint system, you will need to enable the new features through the registration dialog window.

After installing the toolkit:

1. Run the TestPoint editor.
2. If you have TestPoint version 3.3 or earlier, you will also need to open a file that uses the internet toolkit features. Use File/Open and select the TCPIP.TST example from the WEB directory.

**You will get a message about features that have not yet been registered and enabled. This message will give specific instructions on how to enable your software and register it.**

Registering your software can always be done via web, email, or FAX. You'll need information provided on the TestPoint screen about your computer or key ID and software serial number(s), along with your name, company, phone number, and e-mail address.

**Registration via:**

| | |
|---|---|
| **web** | **www.test-point.com** |
| **email** | **registration@test-point.com** |
| **FAX** | **978-663-2626** |

Help is also available at: support@test-point.com

Details of enabling/registration:

To obtain technical support and permanently enable your software, you will need to register at the TestPoint registration data center. You will obtain a software code that you enter into TestPoint to permanently enable your copy.

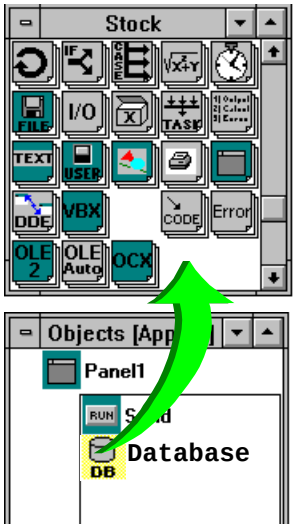
For TestPoint v3.4 and later, enabling your software is done directly through the message window that appears when you run TestPoint. This gets you up and running immediately. You must then send in your registration information to obtain a permanent enabling code.

For TestPoint v3.3 and earlier, a code number is required before your toolkit software is fully enabled. Normally this code is delivered with your software package. If it is not, contact us and we'll get it to you right away.

While your software copy remains unenabled, you will still be able to use the features of the toolkit, but you will not be able to save applications which use these features.

## *Optional - Add the Database object to your Stock window*

You may wish to add the Database object to your Stock window, so it is easy to use in creating new applications. (If not, you can always copy it from DATABASE.TST and paste it into new applications). To add it to the stock, drag it from the Objects window to an **empty** area in the Stock window, hold down the **Ctrl** key, and drop it into the stock:

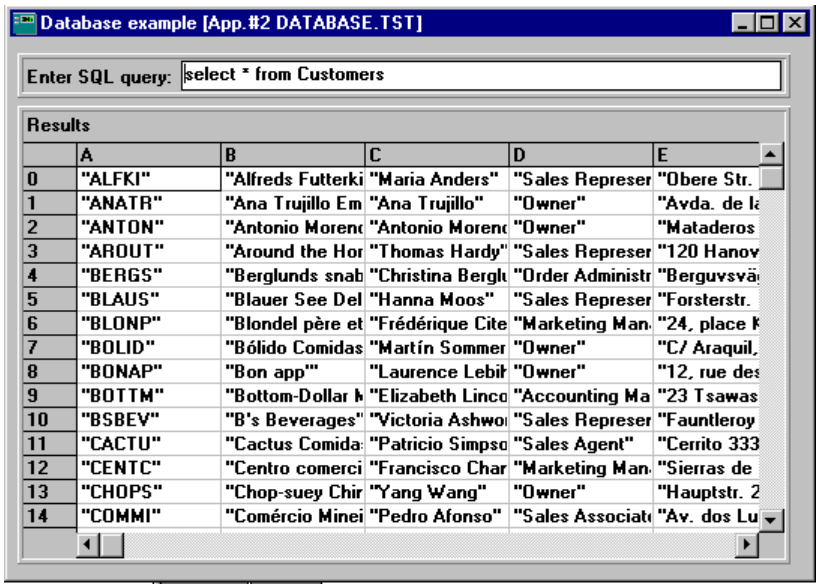



**Make sure to drop on an empty area**

**Hold down the Ctrl key when you drop the object**

# *Chapter 2. Introduction*

The TestPoint Database Toolkit enables your applications to directly interact with all popular databases, such as Microsoft Access, Oracle, SQL Server, and many others.




| | A | B | C | D | E |
|---|---|---|---|---|---|
| 0 | "ALFKI" | "Alfreds Futterki | "Maria Anders" | "Sales Represer | "Obere Str. |
| 1 | "ANATR" | "Ana Trujillo Em | "Ana Trujillo" | "Owner" | "Avda. de la |
| 2 | "ANTON" | "Antonio Moren | "Antonio Moren | "Owner" | "Mataderos |
| 3 | "AROUT" | "Around the Hor | "Thomas Hardy" | "Sales Represer | "120 Hanov |
| 4 | "BERGS" | "Berglunds snab | "Christina Bergl | "Order Administr | "Berguvsvä |
| 5 | "BLAUS" | "Blauer See Del | "Hanna Moos" | "Sales Represer | "Forsterstr. |
| 6 | "BLONP" | "Blondel père et | "Frédérique Cite | "Marketing Man | "24, place K |
| 7 | "BOLID" | "Bólido Comidas | "Martín Sommer | "Owner" | "C/ Araquil, |
| 8 | "BONAP" | "Bon app'" | "Laurence Lebih | "Owner" | "12, rue des |
| 9 | "BOTTM" | "Bottom-Dollar M | "Elizabeth Linco | "Accounting Ma | "23 Tsawas |
| 10 | "BSBEV" | "B's Beverages" | "Victoria Ashwo | "Sales Represer | "Fauntleroy |
| 11 | "CACTU" | "Cactus Comida | "Patricio Simpso | "Sales Agent" | "Cerrito 333 |
| 12 | "CENTC" | "Centro comerci | "Francisco Char | "Marketing Man | "Sierras de |
| 13 | "CHOPS" | "Chop-suey Chir | "Yang Wang" | "Owner" | "Hauptstr. 2 |
| 14 | "COMMI" | "Comércio Minei | "Pedro Afonso" | "Sales Associat | "Av. dos Lu |

You can store information into a database and query and retrieve information, all from your custom TestPoint application. The Database Toolkit contains two new TestPoint objects:

Database Object - used to insert, delete, and select data

Database Utility object - used to get information about a database's structure (its tables and columns)

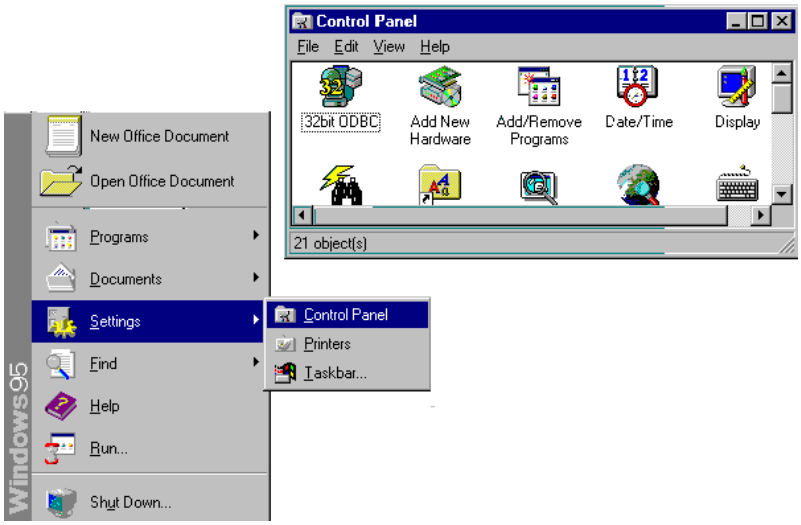
With these objects, you can access any database which supports the industry-standard **ODBC** (Open DataBase Connectivity) drivers. This includes Microsoft Access, Oracle, SQL Server, and many others, including minicomputer and mainframe databases.

# Chapter 3. Setting up ODBC Data Sources

The first step in accessing any database from TestPoint is to configure your data source information in the Microsoft ODBC setup screens.

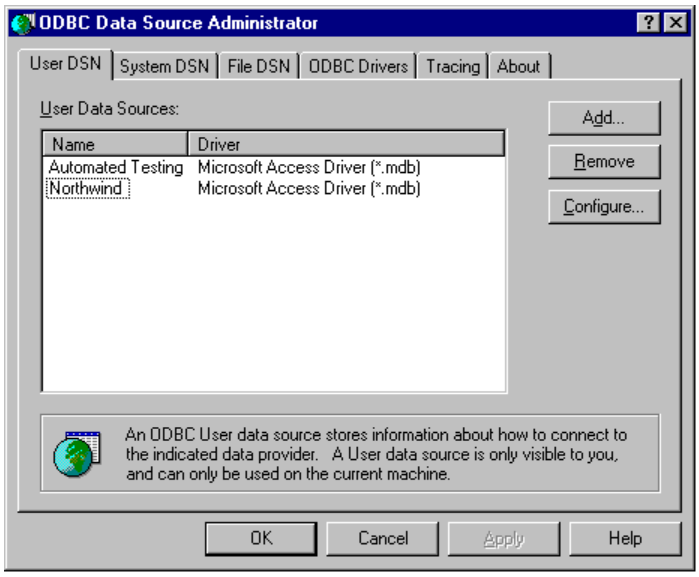
ODBC (Open DataBase Connectivity) is an industry standard interface to databases. Each database program vendor provides drivers and setup screens that comply with this specification.

Begin by opening the Windows Control Panel. In Windows 3.x, you can find this icon in the **Main** program group in Program Manager. In Windows 95, 98, or NT, you use the **Start** button and the **Settings** menu.



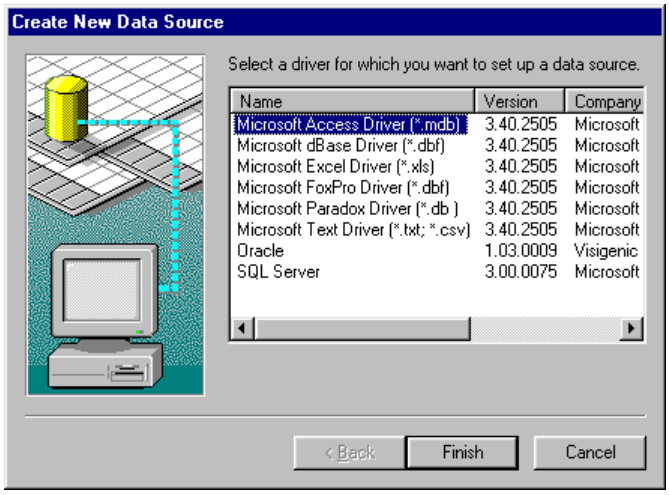

Double-click on the **ODBC** icon in the Control Panel.

(If you do not have an ODBC icon, you may not have installed any databases or ODBC drivers. Go back to the installation for your database and make sure you choose ODBC support if it is an optional part of the installation.)

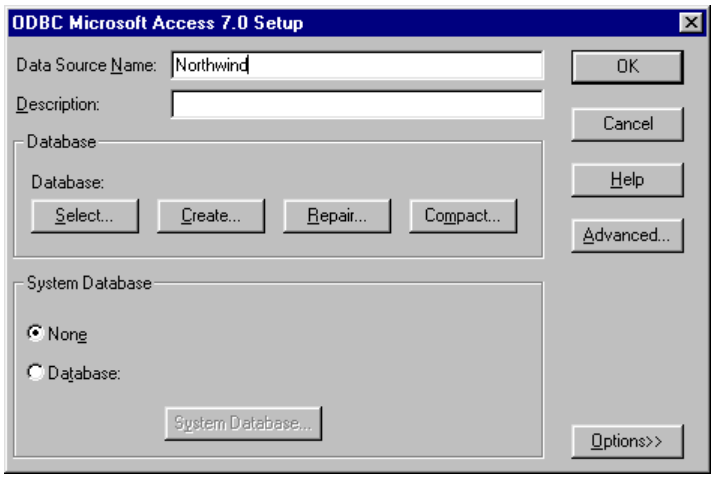



Choose the **Add** button to add a new data source.

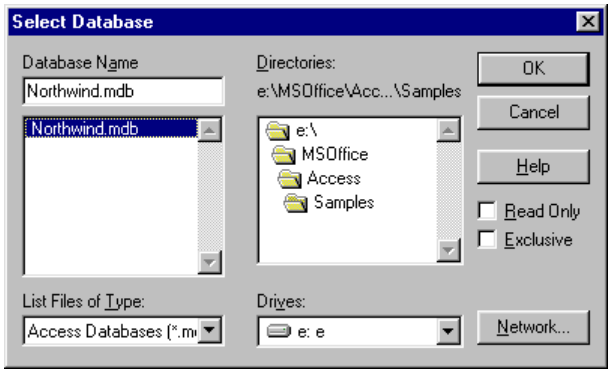
Select the desired database type:



Then, fill in the dialog window with the required information. This will vary depending on the database program. For Microsoft Access, for example, you just need to choose a name for your data source:




You can name your data source any name you like.
Then push the **Select** button and choose the database file:

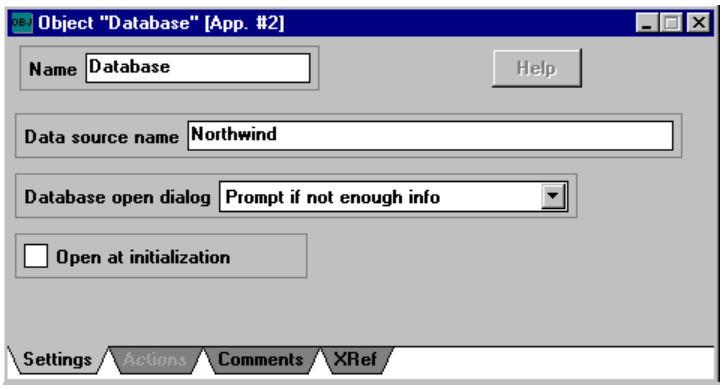



**If you have Microsoft Access, you might want to experiment by creating a data source named "Northwind" connected to the sample Northwind, Inc. database provided with Access.**

# *Chapter 4. Example - Querying a Database*

Open the application **DATABASE.TST** in the DATABASE directory.

Double-click on the database object and enter a data source name that you have configured on your system. If you have Microsoft Access and set up a "Northwind" data source, enter that name.




Put the system into Run mode (using the Mode menu), and enter an SQL query, such as:

SQL

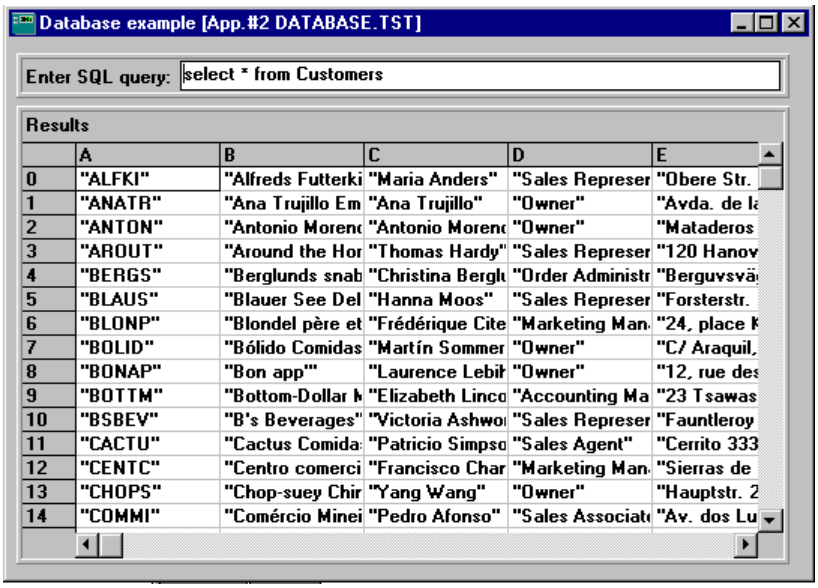
```
SELECT * FROM Customers
```

"Customers" is just an example - you must use a table name that is appropriate for your particular database.

**SELECT** is a keyword in SQL, the industry standard database language, which means to retrieve records that match a given set of conditions. In this case, by using "*", we are retrieving all fields from the chosen table.

A more complete introduction to SQL follows in a later section. Don't worry if this command doesn't make sense to you yet.

The Results grid will fill with up to 100 records of retrieved data.




| | A | B | C | D | E |
|---|---|---|---|---|---|
| 0 | "ALFKI" | "Alfreds Futterki | "Maria Anders" | "Sales Represer | "Obere Str. |
| 1 | "ANATR" | "Ana Trujillo Em | "Ana Trujillo" | "Owner" | "Avda. de la |
| 2 | "ANTON" | "Antonio Moren | "Antonio Moren | "Owner" | "Mataderos |
| 3 | "AROUT" | "Around the Hor | "Thomas Hardy" | "Sales Represer | "120 Hanov |
| 4 | "BERGS" | "Berglunds snab | "Christina Bergl | "Order Administr | "Berguvsvä |
| 5 | "BLAUS" | "Blauer See Del | "Hanna Moos" | "Sales Represer | "Forsterstr. |
| 6 | "BLONP" | "Blondel père et | "Frédérique Cite | "Marketing Man | "24, place K |
| 7 | "BOLID" | "Bólido Comidas | "Martín Sommer | "Owner" | "C/ Araquil, |
| 8 | "BONAP" | "Bon app'" | "Laurence Lebih | "Owner" | "12, rue des |
| 9 | "BOTTM" | "Bottom-Dollar M | "Elizabeth Linco | "Accounting Ma | "23 Tsawas |
| 10 | "BSBEV" | "B's Beverages" | "Victoria Ashwo | "Sales Represer | "Fauntleroy |
| 11 | "CACTU" | "Cactus Comida: | "Patricio Simpso | "Sales Agent" | "Cerrito 333 |
| 12 | "CENTC" | "Centro comerci | "Francisco Char | "Marketing Man | "Sierras de |
| 13 | "CHOPS" | "Chop-suey Chir | "Yang Wang" | "Owner" | "Hauptstr. 2 |
| 14 | "COMMI" | "Comércio Minei | "Pedro Afonso" | "Sales Associat | "Av. dos Lu |

The action list that does this is quite simple:

| | | | |
|---|---|---|---|
| 1) | Prepare query | Database | query-string=Enter SQL query: |
| 2) | Execute query | Database | parameters=_____ |
| 3) | Get result rows | Database | max. rows=100 |
| 4) | Set | Results | to Database |

Line 1 gives the Database object an SQL query statement to process, from the data-entry field labelled "Enter SQL query".
Line 2 executes that statement.
Line 3 gets up to 100 rows (records) of result data. This will be a LIST of VECTORs of STRINGs.
Finally, line 4 displays the results in a Grid object.

# Chapter 5. Tutorial - Databases and SQL

## *Databases, Standards, and TestPoint*

A **database** is a repository of information, together with software for access and retrieval. The information stored in the database may be as simple as a list of names, or it may be complex inter-related sets of data such as customers, orders, products, pricing, and vendors.

The data in a database may be stored in one file or many, on one computer or many, and in almost any format. Each database program on the market has its own method of storage and retrieval. However, industry standards have defined both a language for interacting with databases and a software specification. The universal database language is **SQL**, which stands for **Structured Query Language**. SQL is covered in the next sections of this chapter.

The software specification is **ODBC**, which stands for **Open DataBase Connectivity**. All major database software packages have ODBC drivers available - usually included with the software.

The TestPoint Database Toolkit objects take an SQL command provided by you and pass it on to the ODBC software layer, which in turn passes it on to the database software. The database processes the command and returns an appropriate response or error message.

### Tables

A database normally contains one or more **tables** of data, like the one portrayed below:

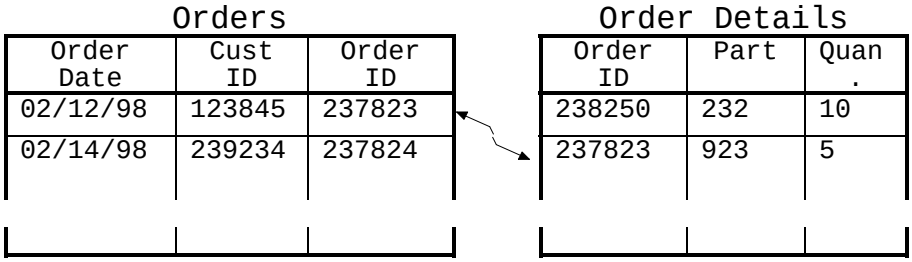
Customers

| Company | Address | Phone | | Country |
|---|---|---|---|---|
| ABC, Inc. | 123 Main St. | 345-3434 | .. . | USA |
| Big Corp. | 1 Big Way | 348-4834 | | USA |

A table consists of **rows** (or records) of one or more **columns** (or fields) each. In the example above, a Customer table has columns for the customer's company name, address, phone number, and so forth.

### *Relations*

Databases often have many tables, with some columns containing common data, to relate the information in the various tables. Most modern databases support this **relational** model.




In the example above, the Orders table stores each separate order that has been placed. The Details table stores each item ordered. The "Order ID" field indicates which detail records relate to which order records. Many detail records can exist for one single order record.

### *Creating Databases*

Databases are usually created, and the table structures defined, using a program provided with the particular database software package being used. For example, Microsoft Access is a screen-oriented program that combines the functions of database creation, updating, querying, and report generation or printing.

It is possible to create new tables, delete or rename tables, and do other major database modifications through the SQL language, using SQL statements like the "CREATE TABLE" command. This can even be done from TestPoint using the Database Toolkit. It is generally recommended, however, that you use the tools provided with the database for major structural changes, and keep backups of your work.

### *Adding, Modifying, and Finding Data*

The main function of a database is not just to store the data, but to provide powerful ways to modify and use the data.

Many database programs like Microsoft Access provide on-screen forms for data entry, that you can customize. These are very useful for creating human-entry applications, but not for storing data produced by a program, such as TestPoint.

To store and retrieve data from TestPoint, the Database Toolkit provides the Database object, which uses SQL commands to interact with a database.

## *SQL - Structured Query Language*

SQL is the industry-standard command language for databases. There are many SQL commands, but the most commonly used are:

SELECT
INSERT
UPDATE
DELETE

Some databases may support only these four commands. Others support a complete set.

**This chapter covers basic information on SQL.**
**Need more information?**
**Try the internet:**
**http://www.pchelponline.com (search on SQL)**
**http://www.metacrawler.com (search on SQL)**

## *SELECT*

The SELECT command is used to search a database table for records that match specified conditions. It returns a group of result rows and columns. The general syntax for SELECT is:


```
SELECT  [DISTINCT]  { * | columnname,... }
        FROM tablename
        [WHERE condition]
        [GROUP BY columname,...]
        [ORDER BY columnname,...]
```

(where square brackets [] denote an optional part of the command, a vertical bar | denotes a choice between alternative values, curly brackets {} are used to group alternatives, and words in italics denote places where you can substitute specific table, column, or function names)

The DISTINCT option means to eliminate duplicate results.

The asterisk "*" instead of a list of column names means to include all the database columns in the result.

Here are some example SELECT statements:


```
SELECT * FROM Customers
```

This example selects all the columns of all the records in the Customers table. That is, the entire table is returned.

```
SELECT CompanyName,CustomerID FROM Customers
    WHERE Country='USA'
```

This example returns only two columns, the company name and customer ID, and only for records where the Country field is 'USA'.

```
SELECT * FROM Products ORDER BY ProductName
```

This example returns all the columns and rows of the Products table, sorted (ordered) by product name.

```
SELECT * FROM Customers WHERE Country=?
```

This example returns all records from Customers for a given country, but the country value is not specified. A question mark is used (known as a "parameter") as a placeholder for a value to be specified later. (See the chapter on parameters).

Note that string constant values are enclosed in single quote characters in SQL.
Question marks can be used as place-holders for values to be supplied when the command is executed (see later chapter on parameters).

This is actually only a partial description of SELECT, which can have many advanced options. Your database documentation will have further information, including a list of the options supported by your particular software.

### *INSERT*

The INSERT command is used to add new records to a database table. The syntax for INSERT is:

SQL

```
INSERT INTO tablename  [(columnname,...)] VALUES
      (value,...)
```

The optional columnnames allow the ability to provide values for only selected columns of the new record. In this case, the other columns receive their default values (if defined).

INSERT statements do not return any result rows.

Here are some example INSERT statements:

SQL

```
INSERT INTO Shippers (CompanyName,Phone)
      VALUES ('XYZ Trucking','555-555-5555')
```

This example inserts a new record into the Shippers table, filling in the two named fields with the given string values. There is one other field in this particular table, and it receives a default value.

```
INSERT INTO Shippers VALUES (?,?,?)
```

This example inserts a record into the Shippers table, where all three field values come from parameters to be supplied later. (See the chapter on parameters).

### *UPDATE*

The UPDATE command is used to change one or more values in existing records of a database table. The syntax for UPDATE is:


```
UPDATE tablename SET columnname=value, ...
    [WHERE condition]
```

An UPDATE command can modify one record, or more than one - even all the records in the database.

UPDATE statements do not return any result rows.

Here are some example UPDATE statements:


```
UPDATE Products SET ReorderLevel=10
    WHERE ReorderLevel<10

UPDATE Customers SET CompanyName='BigCorp.
    (subsidiary)'
    WHERE CompanyName="SmallCorp.'
```

### *DELETE*

The DELETE command is used to get rid of existing records in a database table. The syntax for DELETE is:


```
DELETE FROM tablename WHERE condition
```

DELETE statements do not return any result rows.

Here are some example DELETE statements:

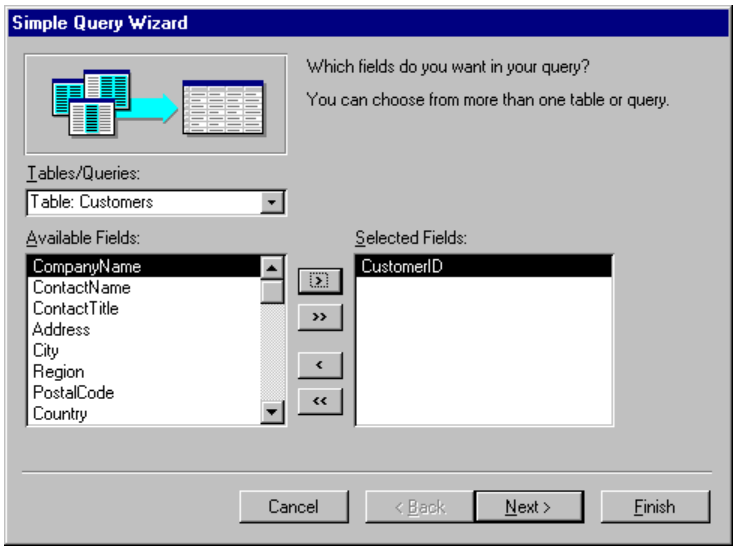

```
DELETE FROM Products WHERE ReorderLevel=0

DELETE FROM Customers WHERE CompanyName='ABC
      Co.'
```

## *Letting the Database program write SQL statements for you*

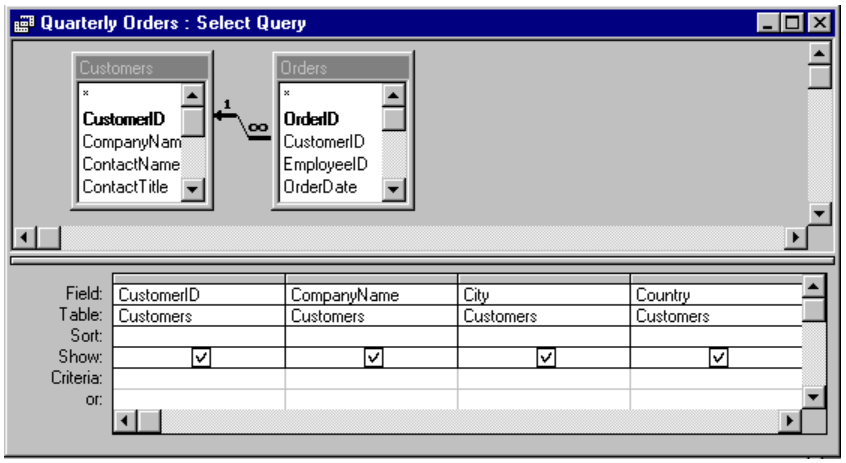
Some databases have an interface that will help you write SQL statements. For example, Microsoft Access lets you fill in a table or form to create a query, and then you can choose the View/SQL menu command to see the same query in the SQL language. Many other database packages also include a "query by example" utility.

### *Microsoft Access*

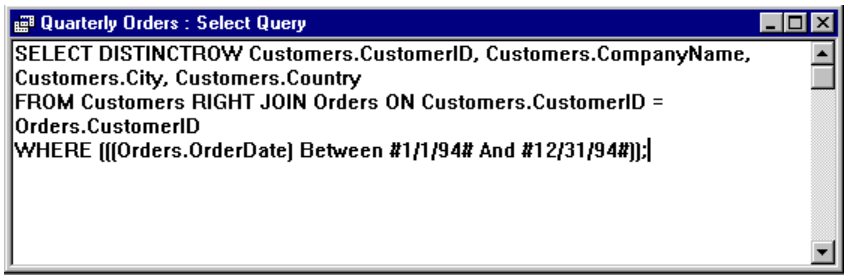
Start by creating a query using Access' tools, such as the Query Wizard:



Once you have a query:




Just select the View/SQL menu command, and you'll see the equivalent in the SQL language. This statement can be directly used with the TestPoint Database Toolkit:

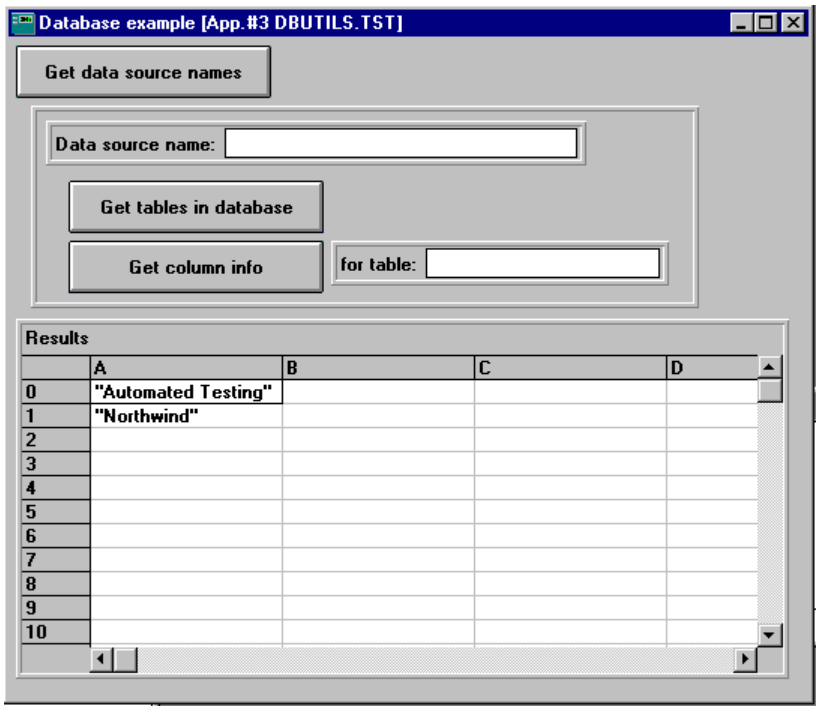


# Chapter 6. TestPoint Database Utilities

The file **DBUTILS.TST** in the DATABASE directory contains a Database Utilities object which allows you to retrieve useful information about the available databases on your system.

## Viewing Data Source Names

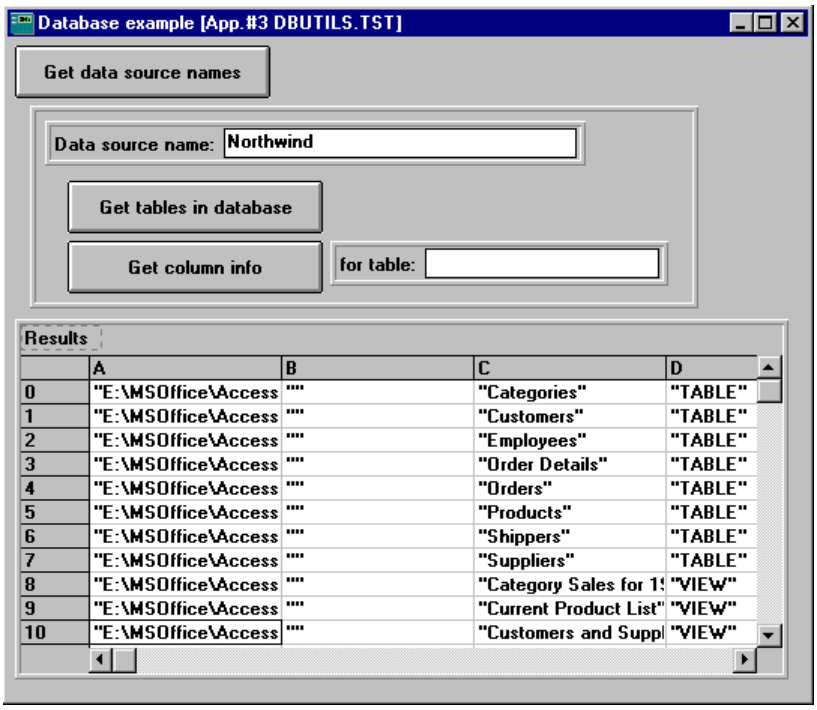
Load DBUTILS.TST into TestPoint, switch to Run mode, and press the **Get data source names** button. This will list all the available data sources you have configured for ODBC.




If no data sources are listed, go back to Chapter 3 and check your database setup.

## *Viewing Tables in a Database*

Next, you can enter one of the data source names and push **Get tables in database**. This will list all the table names.

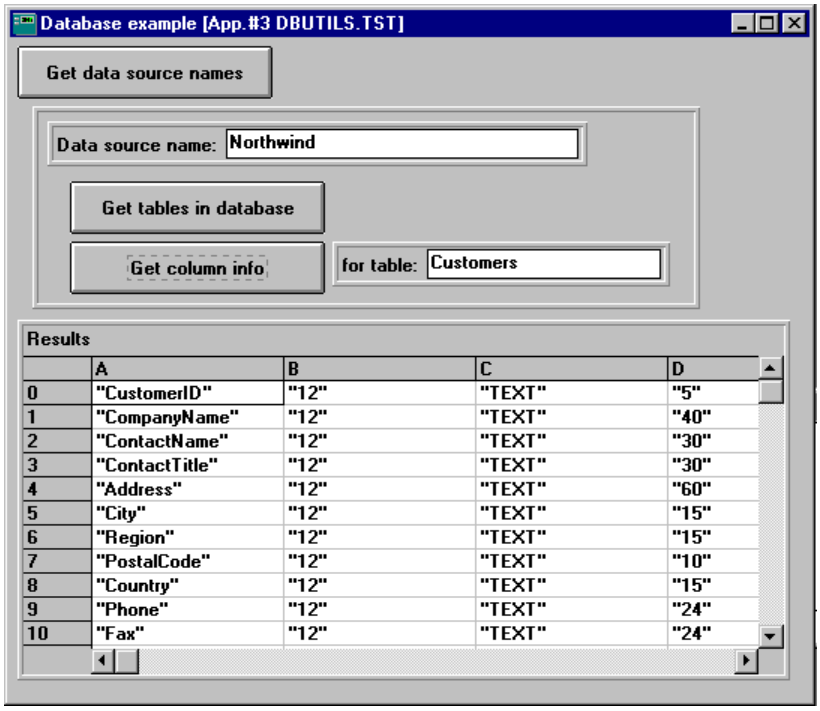



| | A | B | C | D |
|---|---|---|---|---|
| 0 | "E:\MSOffice\Access | "" | "Categories" | "TABLE" |
| 1 | "E:\MSOffice\Access | "" | "Customers" | "TABLE" |
| 2 | "E:\MSOffice\Access | "" | "Employees" | "TABLE" |
| 3 | "E:\MSOffice\Access | "" | "Order Details" | "TABLE" |
| 4 | "E:\MSOffice\Access | "" | "Orders" | "TABLE" |
| 5 | "E:\MSOffice\Access | "" | "Products" | "TABLE" |
| 6 | "E:\MSOffice\Access | "" | "Shippers" | "TABLE" |
| 7 | "E:\MSOffice\Access | "" | "Suppliers" | "TABLE" |
| 8 | "E:\MSOffice\Access | "" | "Category Sales for 1 | "VIEW" |
| 9 | "E:\MSOffice\Access | "" | "Current Product List" | "VIEW" |
| 10 | "E:\MSOffice\Access | "" | "Customers and Suppl | "VIEW" |

**NOTE: Not all databases or ODBC driver versions support this feature (Some versions of the Microsoft Access 95 ODBC driver being prominent examples).**

## Viewing Column (Field) Information

Finally, you can retrieve information about the fields and data types in a particular table by entering the table name and pushing the **Get column info** button.




| | A | B | C | D |
|---|---|---|---|---|
| 0 | "CustomerID" | "12" | "TEXT" | "5" |
| 1 | "CompanyName" | "12" | "TEXT" | "40" |
| 2 | "ContactName" | "12" | "TEXT" | "30" |
| 3 | "ContactTitle" | "12" | "TEXT" | "30" |
| 4 | "Address" | "12" | "TEXT" | "60" |
| 5 | "City" | "12" | "TEXT" | "15" |
| 6 | "Region" | "12" | "TEXT" | "15" |
| 7 | "PostalCode" | "12" | "TEXT" | "10" |
| 8 | "Country" | "12" | "TEXT" | "15" |
| 9 | "Phone" | "12" | "TEXT" | "24" |
| 10 | "Fax" | "12" | "TEXT" | "24" |

This shows the column names, data types, lengths, and other information. Different databases return slightly different information.

# *Chapter 7. The Database Object*

# Settings

## Data source name

This setting provides the information needed to find and open the database.

It can be just the data source name chosen in the ODBC source setup, or it can also include additional user-id and password information. Some databases provide security based on the user identification. You can provide a user ID by using the keyword UID=, as shown in the examples below. If your database requires a password, you can use the keyword PWD=, as shown below.

Here are some examples of appropriate settings for this field, for databases with and without password protection:

```
Sample Database
DatabaseWithPassword;PWD=password
SecureDatabase;UID=admin;PWD=password
```

Note that optional fields like UID and PWD are separated by semicolons.

## Database open dialog

If the database cannot be opened from the information given in the data source setting (for example, a password is required and none was specified in this field), the ODBC drivers can optionally display a dialog window and prompt the user for this information.
This setting lets you configure whether such a dialog will appear.
If set to **Never Prompt**, then no dialog window will appear, and the database open will fail if not enough information is provided.
If set to **Prompt if not enough info**, a dialog window will appear only if more information is required.

### *Open at initialization*

Opening the database may require a noticeable amount of time, depending on the system and the database program. This setting lets you choose whether to open the database immediately during application initialization (when you go into Run mode), or only at the first database action.
If this checkbox is set, the database will be opened at initialization time.

## Actions

### Open

This opens the database, if it is not already open.
Note that this action is not required - if you do a Prepare query command when the database is not yet open, it will be opened automatically.

### Close

This closes the database, if it is open. Note that the database is automatically closed when the TestPoint application terminates (or you go to Edit mode in the editor). However, it may be useful to close the database during execution in some applications.

### Prepare query

This is the first action used in actual interacting with the database. It takes an SQL command string and passes it to the database, but does not yet execute the command.
The command is checked by the database program (not by TestPoint) for validity, and any errors will be reported.

The reason this action is separate from the Execute query action is that you may wish to execute a single SQL statement many times (for example, a statement that inserts a new record into the database). By executing the Prepare action only once, your actions will execute more quickly.

**Note that if you do another Prepare query on the same database object, the first query and its results are no longer available. However, there is no restriction on having multiple database objects accessing the same data source, if you have multiple SQL commands you wish to have ready for execution, or multiple result tables you wish to have available at the same time. See the examples chapter.**

### *Execute query*

This action carries out, or executes, the already prepared SQL command (from the Prepare query action).

If the SQL command retrieves any data (a SELECT command), the results can be accessed using the Get result rows action.

If your SQL query contained parameters, indicated by "?" characters, you can pass in the values for these parameters in this action.
See Chapter 9 for more information on parameters.
The parameters must be passed as a **LIST** datatype. This is most easily done by storing the values into a container. (See the chapter on parameters and the examples following that chapter).

### *Get result rows*

This action retrieves results from the most recently executed query, and returns them as the data value of the database object.

You can retrieve any desired number of rows, or records in a single action.
If you retrieve one row, you will get a TestPoint **LIST** datatype as a result, with each item in the list corresponding to a field in the result table.
If you retrieve more than one row, you will get a **LIST of VECTORS** result datatype.
In all cases, results are returned as TestPoint **STRING**s, regardless of the database field type (string, numeric, currency, date, time, etc.).

After you have retrieved some results, you can use this action again to retrieve additional result rows if they are available.
When no more data is available, the value of the database object will be **NoData** (see Examples).

If you need help in extracting individual strings from this result information, check the Math chapter in the TestPoint Techniques and Reference manual. It has a subsection on manipulating vectors, lists, and arrays. The Math object and the functions **select()**, **index()**, **sublist()**, and **subarray()** can be useful.

# Chapter 8. Examples

## *Query and Display Results*

The example **DATABASE.TST** shows how to do a simple SQL query and display the results in a Grid object. Here is an action list with a specific query string:

| | | | |
|---|---|---|---|
| 1) | Prepare query | Database | query-string="select * from Customers" |
| 2) | Execute query | Database | parameters=_____ |
| 3) | Get result rows | Database | max. rows=100 |
| 4) | Set | Results | to Database |

Note that:

- The "execute query" action does not pass any parameters - they are not used in this example. Parameters are described in the next chapters.
- Up to 100 result rows are retrieved in a single action. The "max. rows" parameter can be any desired value, including 1. In this case, the result data value will be a **LIST of VECTORS of STRING**, because more than one row matches the query, and the database has multiple fields.
- The query string here is a constant entered right into the action line. It can just as easily be an object like a data-entry field, where the user can type in a query.

The query string can also be built in the action list, based on constants and variables both, using a Math object, as in this example:

```
1)  Calculate        Query       with country=Enter country:
2)  Prepare query    Database    query-string=Query
3)  Execute query    Database    parameters=_____
4)  Get result rows  Database    max. rows=100
5)  Set              Results     to Database
```

Here the object "Enter country:" is a data-entry field, and the Math object "Query" has a formula of:

```
"select * from Customers where Country=" & chr(39) & country &
                                        chr(39)
```

Character code 39 decimal is a single-quote character, used around string values in SQL statements. So if the value of the "Enter country" object is the string "USA", the result of the math calculation will be this string:

```
select * from Customers where Country='USA'
```

## *Query and Retrieve Single Values in a Loop*

Sometimes it is useful to retrieve records one at a time for processing, instead of in one large list of vectors. To do this, just use 1 as the "max. rows" value in "Get result rows", and use a loop. This example is on disk as **ROWBYROW.TST** in the DATABASE subdirectory:

```
1)  Prepare query     Database     query-string="select
                                   CompanyName from
                                   Customers"
2)  Execute query     Database     parameters=____
3)  Repeat loop       Loop1        until  Done is true
4)    Get result rows Database     max. rows=1
5)    If/Then/Else    Done         with data=Database
6)      Set           Display      to "end of records"
7)    Else if not     Done
8)      Set           Display      to Database
9)    End if          Done
10) End               Loop1
```

Note that the "Get result rows" action returns an object value of **NoData** when no more matching records exist. The formula for the conditional object "Done" is:

```
type(data)=0
```

This means that Done becomes **true** when the "Get result rows" action reaches the end of the records. In the example, this executes line 6, setting the display to "end of records", and also exits the loop.

## *Using Two or More Database Objects at Once*

A single database object represents a specific query and its results. You can use a single database object to execute one query after another, but if you ever need to have two different queries and their results available at the same time, you will need more than one Database object.

The example **TWOQUERY.TST** shows how this can be done. There is no restriction against having more than one Database object accessing the same database at the same time.

```
1)  Prepare query     DBCustomers  query-string="select
                                   CustomerID, CompanyName
                                   from Customers"
2)  Execute query     DBCustomers  parameters=____
3)  Repeat loop       Loop1        until Done is true
4)    Get result rows DBCustomers  max. rows=1
5)    If/Then/Else    Done         with data=DBCustomers
6)      Set           Customer     to "--- done ---"
7)      Clear         Orders
8)    Else if not     Done
9)      Calculate     SelectedItem with data=DBCustomers item=1
10)     Set           Customer     to SelectedItem
11)     Calculate     SelectedItem with data=DBCustomers item=0
12)     Calculate     OrderQuery   with name=SelectedItem
13)     Prepare query DBOrders     query-string=OrderQuery
14)     Execute query DBOrders     parameters=____
15)     Get result rows DBOrders   max. rows=100
16)     Set           Orders       to DBOrders
17)   End if          Done
18) End               Loop1
```

**Notes:**

- In the first query, using object "DBCustomers", we retrieve both the "CustomerID" and "CompanyName" fields. The company name will be used for screen display, and the customer ID will be used in querying for that customer's orders.
- The Math object "SelectedItem" is used to pick out the ID or the company name from the customer query. Its formula setting is:

  select(data,item)

  because the data type of a single result row is a **LIST**, so select() is the appropriate math function.
- The Math object "OrderQuery" creates a query string using this formula:

  "select * from Orders where CustomerID=" & chr(39) &
  name & chr(39)
- The loop and "Done" object are the same as in the previous example.

You can also use multiple Database objects when you need to work with more than one data source.

# *Chapter 9. Using Parameters to Pass Data into the Database*

## SQL Parameter Syntax

So far, the examples have used a complete SQL statement in the "Prepare query" action. In some cases, a SELECT query can be built using a Math object, combining variable values with string constants.

However, there is another way to use variable values in an SQL statement: parameters. When you have a value to put into an SQL statement, instead of placing the actual value in the statement directly, you may use a question-mark character instead. For example:


```
SELECT * from Customers where Country=?
```

This methods lets you prepare the query once, and then execute it multiple times with different values. Since preparing the query can sometimes be a time-consuming operation, parameters can be very useful.

Here are other examples of the use of parameters:


```
INSERT INTO Shippers (CompanyName,Phone) VALUES (?,?)

SELECT * from Customers where Country=? and
      CompanyName like ?

UPDATE Products SET ReorderLevel=? WHERE
      ProductName='Tofu'
```

Parameters are particularly useful in INSERT statements, where the data is typically variable values, and it is often required to insert records repeatedly and quickly during program execution.

## *Using SQL Parameters in TestPoint*

With TestPoint's Database object, you pass parameters in the "Execute query" action.

If there is only one parameter you can put it directly on the action line:

```
2)  Execute query   Database    parameters="A parameter
                                value"
```

If you have more than one parameter (more than one question mark in the SQL statement), you must pass a **LIST** data type. The easiest way to build a list of parameter values is usually the Container object and the "Store into" action:

```
2)  Store into      Container   from "Parameter1", "Another",
                                "Last one"
3)  Execute query   Database    parameters=Container
```

## *TestPoint and SQL Data Types*

Databases can store many different types of data such as text, numbers, currency, date, time, binary, boolean, and more.

TestPoint has number, string, vector, array, and list data types.

Normally, TestPoint uses **STRING** data for all interaction with databases. All result fields are retrieved as strings. All parameters are normally passed in as strings (although some databases provide the ability for TestPoint to determine the desired data type of parameter, and TestPoint will use a numeric type if the database specifies this).

The database drivers (ODBC) normally convert between the various types automatically. Most of the conversions are obvious. All conversions should be documented in the database manuals and help files. Some conversions, such as date/time, may depend on system settings for a format decision (such as mm/dd/yyy or dd/mm/yyyy order for dates).

So, for example, you can pass in a parameter string value of:

"10:34"

as a string, for a field which in the database is a Time data type.

### *Numeric Fields in Microsoft Access*

**One important exception to automatic conversions is the Microsoft Access database.** The drivers for Access do not have the capability of automatically converting string parameters to numeric values **in "WHERE" clauses of SELECT statements**, so if you execute a query like this one:


```
SELECT * from Products where UnitsInStock < ?
```

you will get an error in the parameter type.
The solution, in the case of Microsoft Access, is to use the built-in function VAL(), which converts strings to numbers, like this:


```
SELECT * from Products where UnitsInStock < VAL(?)
```

This statement is very specific to Microsoft Access, since the VAL() function is part of Access' built-in BASIC language, not part of the SQL standard.

# Chapter 10. More Examples

# *Inserting Records Into a Database*

In testing and data acquisition applications such as those you may create using TestPoint, it is often required to keep a log of measurement results, and a database is a great tool for this. By inserting new records into the database for each new test result, you can bring all the database tools for search, retrieval, statistics and reporting to bear on your results.

To insert records containing variable data, just use parameters and the "INSERT" statement of SQL. You can prepare the statement just once, and then execute it for each new record to be inserted:

```
1)  Prepare query    Database      query-string="INSERT into
                                   Results VALUES (?,?,?,?)
2)  Do loop          TestingLoop   while  notDone is true
3)                   ... etc. ...
... etc. ...
7)     Store into    Container     from  Voltmeter1, Calc2,
                                   Value3, SerialNumber
8)     Execute query Database      parameters=Container
9)  End loop         TestingLoop
```

The INSERT statement requires a table name to insert into, optionally a list of column (field) names, and a list of values, which can be parameters:


INSERT INTO *tablename* (*columnname*,...) VALUES (?,?...)

Here are some example INSERT statements, some with parameters:

```
INSERT INTO Products (ProductName) VALUES (?)

INSERT INTO Shippers VALUES (9999,'XYZ Trucking',
        '685-343-3333')

INSERT INTO Shippers (CompanyName,Phone) VALUES (?,?)
```

## *Queries with Variable Parameters*

It is often useful to execute a query to search for database records, where some of the search criteria will be variables entered by the user or calculated from other information.

This can easily be done by using parameters in a SELECT statement:

```
1)  Prepare query    Database    query-string="select * from
                                 Results where Type=?"
2)  Execute query    Database    parameters=Entry-field-for-type
3)  Get result rows  Database    max. rows=100
```

In this example, the SELECT query searches for records where the "Type" field matches a parameter supplied during the Execute action, from a data-entry field on the screen.

# *Chapter 11. Notes on Specific Database Programs*

# Microsoft Access

## Passwords

Microsoft Access does allow you to assign password protection for a database. If you do so, you can either let the user get a dialog box when your application first opens the database, or specify the password in the "Data source name" setting, as in this example:

```
MyDatabase;PWD=password
```

## Data Type Conversions

Access is not generally able to automatically convert data types from string to numeric for parameters in a WHERE clause of a SELECT statement. So if you execute a query like this one:

```
SELECT * from Products where UnitsInStock < ?
```

and "UnitsInStock" is numeric, you will get an error in the parameter type.
The solution, in the case of Microsoft Access, is to use the built-in function VAL(), which converts strings to numbers, like this:

```
SELECT * from Products where UnitsInStock < VAL(?)
```

## Utiltity Function Limitations

If you run DBUTILS.TST and use the Database Utilities object, you may find that Access does not support the "Get table names" function. (we tested with Microsoft Access 95 - version 7.00, the feature depends on the ODBC driver revision level).

### Passwords

SQL Server generally provides user and password protection for a database. If you do so, you can either let the user get a dialog box when your application first opens the database, or specify the user ID and password in the "Data source name" setting, as in this example:

MyDatabase;UID=admin;PWD=password

### Starting the service

The SQL server database service must be started in order to successfully open a database. Check the SQL Server documentation to see how to manually start the service or set it up to start automatically.

In a client/server network, you must also have the database client software correctly installed and configured to access your database server.

### Passwords

Oracle generally provides user and password protection for a database. If you do so, you can either let the user get a dialog box when your application first opens the database, or specify the user ID and password in the "Data source name" setting, as in this example:

MyDatabase;UID=sa;PWD=password

# Chapter 12. Distributing Runtimes

## *Licensing Policy*

No runtime fees are required to distribute applications written using TestPoint or the TestPoint Database Toolkit. No execution key is required on the printer port for runtimes.

You may freely copy and distribute the runtime support files, including the files TPODBC.DLL, TPODBC16.EXE, and TPODBC32.EXE with your applications.

You may not distribute the TestPoint editor or other development-only portions of the software to anyone else.

## *Creating a runtime - Files to distribute*

To create a runtime distribution, use the standard Utilities menu command Make Runtime Disk, just as you would for any other TestPoint application.

Use the **Add Files** button to add the following files, which are **required** for Database applications:

TPODBC.DLL
TPODBC16.EXE
TPODBC32.EXE
MFC42.DLL
MSVCRT.DLL

You will find the last two files in your **windows\system** directory (in Windows NT, this is often named **winnt\system32**).

## *Registering the database servers*

After packaging the runtime, you need one more step, which requires manually editing the file **SETUP.INF** placed on your runtime distribution disk.

Add the following lines to the first section marked **[Application]**:

```
run=TPODBC16.EXE
run32=TPODBC32.EXE
```

Add this line (containing three asterisk characters) to the section beginning with **[Dirs]**:

```
9=***
```

And finally, modify the lines in the **[Files]** section for the files MFC42.DLL and MSVCRT.DLL as follows:

```
Application Name, MFC42.DLL, 9,Y,N,O
Application Name, MSVCRT.DLL, 9,Y,N,O
```

**(those are letter O's at the end of each line, not zeroes)**.

These lines are required to ensure that during SETUP, these database server applications are executed and registered with the operating system on the newly installed computer.

TPODBC16 and TPODBC32 are ActiveX server applications which facilitate the interface between TestPoint and the ODBC drivers. They do not display any windows on the screen, but run in background to provide database services. The 16-bit version is for Win3.x, and the 32-bit version for Win95, 98, or NT. TestPoint automatically selects the one appropriate for the version of Windows where the application is executing.

# *Chapter 13. Troubleshooting*

### 25000 - Cannot open database server

This message appears when the Database object is unable to run the required ActiveX database server application (TPODBC16 or TPODBC32, depending on your Windows version). These servers are automatically installed and registered when you install the Toolkit. If you need to re-register them, just run the program once, and it will be registered with the operating system.

### 25000 - ... other messages ...

You can also get error 25000 when you cannot open the database for other reasons. In this case the specific message from the ODBC drivers will be displayed.

### 25001 - .... other messages ...

**Error preparing SQL statement**. This error occurs if the ODBC drivers return an error in the Prepare action. The ODBC error string will be displayed.

### 25002 - Database not opened...
### Database open, but no valid query specified ...

This error occurs on "Execute query" if a required earlier step of opening the database or preparing the query was skipped, or had an error.

### 25003 - ... other messages ...

**Error executing SQL statement**. This error occurs if the ODBC drivers return an error in the Execute action. The ODBC error string will be displayed.

### *25004 - ... other messages ...*

**Error during result retrieval**. This error occurs if the ODBC drivers return an error in the Get result rows action. The ODBC error string will be displayed.

### *25005 - Error in parameter type*

This error can occur in the Execute query action if the data type of a parameter does not match the required data type, and the database does not automatically convert types. **In Microsoft Access 95, you may have to use the VAL(?) syntax for a parameter that corresponds to a numeric field.**

**A**

Access, 9-5, 11-2

**C**

Close, 7-4

Code number, 1-3

Column, 5-2

Column info, 6-4

Control Panel, 3-2

Conversion, 9-4

Copy/paste, 1-5

**D**

Data source, 3-2, 6-2, 7-2

Data type, 5-2, 7-6, 9-4, 9-5, 11-2

Database object, 7-2

Date/time, 9-4

DELETE, 5-11

Dialog, 7-2

Distribution, 12-2

**E**

Enabling, 1-3

End of data, 8-4

Error message, 13-2

Examples, 8-2

Execute query, 7-5

**F**

Features, 2-2

Field, 5-2

Field info, 6-4

Files, 12-3

**G**

Get result rows, 7-6

**I**

Initialization, 7-3

INSERT, 5-2, 5-9, 10-2

Installation, 1-2

**K**

Key, 1-3, 12-2

**L**

License, 12-2

Loop, 8-4

**M**

Microsoft Access, 9-5, 11-2

Microsoft SQL Server, 11-3

**O**

ODBC, 2-2, 3-2

Open, 7-2, 7-4

Oracle, 11-4

**P**

Parameter, 9-2, 10-2

Password, 7-2, 11-2, 11-3, 11-4

Prepare query, 7-4

**Q**

Query, 8-2

**R**

Record, 5-2

Register, 12-4

Row, 5-2

Runtime, 12-2, 12-4

**S**

SELECT, 5-2, 5-7

Serial number, 1-3

Server, 12-4

Settings, 7-2

Setup, 1-2, 12-4

SETUP.INF, 12-4

SQL, 5-2, 7-4, 9-2

SQL Server, 11-3

Stock, 1-5

System requirements, 1-2

**T**

Table, 5-2

Table names, 6-3

TPODBC.DLL, 12-3

TPODBC16.EXE, 12-3, 12-4

TPODBC32.EXE, 12-3, 12-4

Two objects, 8-5

TWOQUERY, 8-5

**U**

UPDATE, 5-2, 5-10

User ID, 7-2

Utilities, 6-1

**V**

VAL, 9-5