Delphi Advanced Programming Technology

CHAPTER10 USING SQL IN APPLICATIONS

Professor Zhaoyun Sun

Introduction

- SQL (Structured Query Language) is an industry-standard language for database operations.

- Delphi enables your application to use SQL syntax directly through the *TQuery* component.

10.1 Using TQuery

- *TQuery* is a dataset component, and shares many characteristics with *TTable*, as described in Chapter 9, “Using data access components and tools.”

- In addition, *TQuery* enables Delphi applications to issue SQL statements to a database engine.

When to use TQuery

- For simple database operations, ***Ttable*** is often sufficient and provides portable database access through the **BDE**.

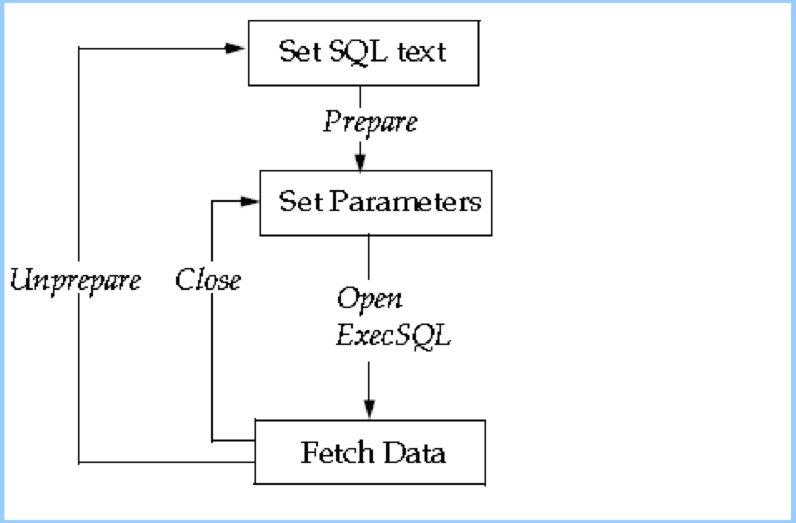
- However, ***TQuery*** provides additional capabilities that ***TTable*** does not. Use ***TQuery*** for:

When to use TQuery

- Multi-table queries (joins).
- Complex queries that require sub-SELECTs.
- Operations that require explicit SQL syntax.

How to use TQuery

TQuery methods and flow



Note ***Prepare*** **applies only to dynamic queries. It is not required, but is recommended in most cases.**

The SQL property

- The ***SQL*** property contains the text of the SQL statement to be executed by a Query component.

- This property is of type ***TStrings***, which means that it is a series of strings in a list. The list acts very much as if it were an array, but it is actually a special class with unique capabilities.

The SQL property

- A Query component can execute two kinds of SQL statements:
  - – Dynamic SQL statements
  - – Static SQL statements

The SQL property

```
SELECT * FROM CUSTOMER WHERE CUST_NO = 1234
```

- A *dynamic* SQL statement, also called a *parameterized* statement, includes parameters for column or table names. For example, this is a dynamic SQL statement:

```
SELECT * FROM CUSTOMER WHERE CUST_NO = :Number
```

- The variable *Number*, indicated by the leading colon, is a parameter which must be provided at run time and may vary each time the statement is executed.

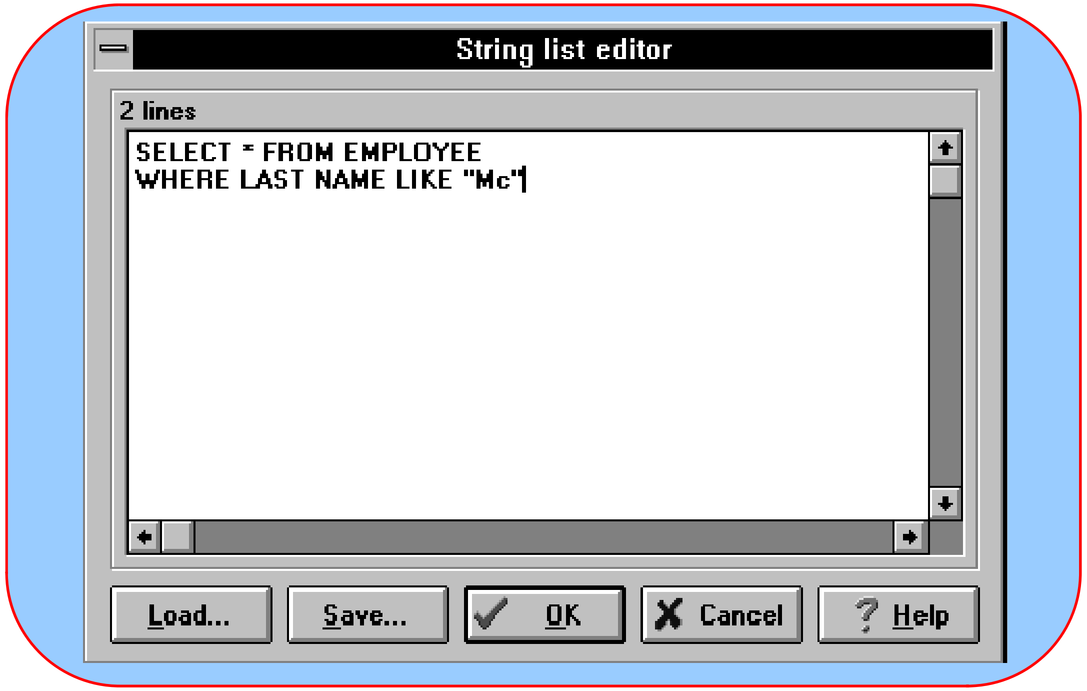
Creating the query text

- You can enter the SQL text for a ***TQuery*** at design time by double-clicking on the ***SQL*** property in the Object Inspector, or choosing the ellipsis button.

- The String List Editor opens, enabling you to enter an SQL statement.

Creating the query text

Editing SQL statements in the String List Editor



```
SELECT * FROM EMPLOYEE
WHERE LAST NAME LIKE "Mc"
```

Creating the query text

- To specify SQL text at run time, an application should first close the query with *Close* and clear the *SQL* property with *Clear*. For example,

```
Query1.Close; {This closes the query}
Query1.SQL.Clear; {This clears the
contents of the SQL property}
```

Creating the query text

- It is always safe to call *Close*—if the query is already closed, the call will have no effect. Use the *SQL* property's *Add* method to add the SQL statements to it. For example,

```
Query1.SQL.Add('SELECT * FROM COUNTRY');
Query1.SQL.Add('WHERE NAME = ''ARGENTINA''');
```

Creating the query text

- You can also use the *LoadFromFile* method to assign the text in an SQL script file to the *SQL* property. For example,

```
Query1.SQL.LoadFromFile('C:\MYQUERY.TXT');
```

Using the Visual Query Builder

Working in the Visual Query Builder

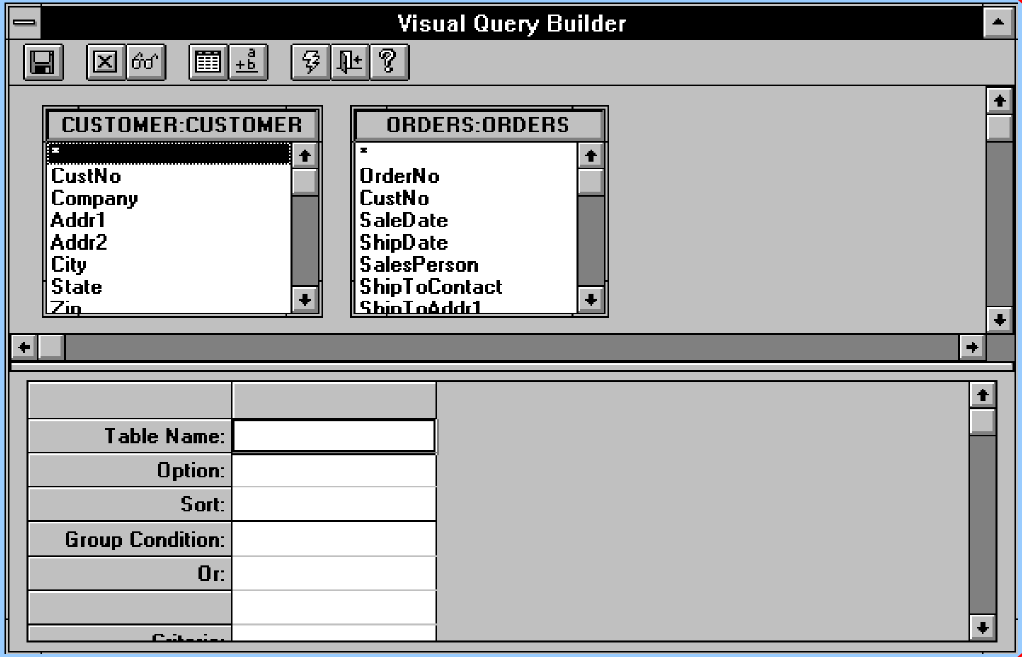


Table Name:	
Option:	
Sort:	
Group Condition:	
Or:	
Criteria:	

Executing a query

- At design time, you can execute a query by changing its *Active* property in the Object Inspector to True.

- The results of the query will be displayed in any data controls connected to the Query component (through a data source).

Executing a query

- At run time, an application can execute a query with either the ***Open*** or the ***ExecSQL*** methods. Use *Open* for SQL statements that return a result set (SELECT statements). Use ***ExecSQL*** for all other SQL statements (INSERT, UPDATE, DELETE, and so on). For example,

```
Query1.Open; {Returns a result set}
```

Executing a query

- If the SQL statement does not return a cursor and a result set from the database, use *ExecSQL* instead of *Open*. For example,

```
Query1.ExecSQL; {Does not return a result set}
```

- If you don't know at design time whether a query will return a result set, use a try...except block with Open in the try part and ExecSQL in the except part.

Getting a live result set

- A *TTable* component always returns a live result set to an application.

- That is, the user sees the data “live” from the database, and can make changes to it directly through data controls.

- A *TQuery* can return two kinds of result sets:

Getting a live result set

– "Live" result sets: As with *TTable* components, users can edit data in the result set with data controls. The changes are sent to the database when a *Post* occurs, or when the user tabs off a control.

— "Read only" result sets: Users cannot edit data in the result set with data controls

Getting a live result set

- By default, a query always returns a read-only result set.

- To get a live result set, an application must request it by setting the *RequestLive* property of *TQuery* to True.

Getting a live result set

- However, for the BDE to be able to return a live result set, the SELECT syntax of the query must conform to the guidelines given below.

- If an application requests a live result set, but the syntax does not conform to the requirements, the BDE returns a readonly result set (for local SQL) or an error return code (for passthrough SQL).

- If a query returns a live result set, Delphi will set the *CanModify* property to True.

Getting a live result set

Types of query result sets

RequestLive set	CanModify	Type of result
FALSE	FALSE	Read-only result set
True—SELECT syntax meets requirements	TRUE	Live result set
True—SELECT syntax does not meet requirements	TRUE	Read-only result set

10.2 Dynamic SQL statements

- A dynamic SQL statement (also called a parameterized query) contains parameters that can vary at run time.

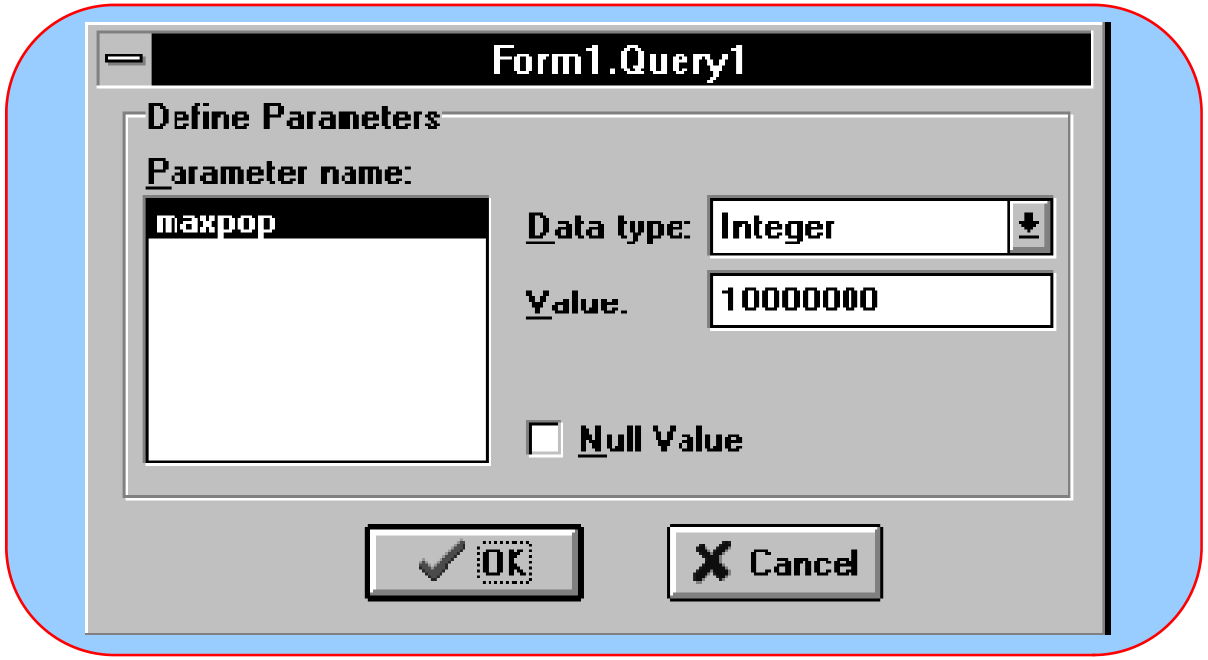
Supplying values to parameters

- At design time, you can supply values to parameters with the Parameters Editor.

- Invoke the Parameters Editor by selecting a *TQuery* component, right-clicking the mouse, and then selecting Parameters Editor.

Supplying values to parameters

Parameters Editor


Form1.Query1

Define Parameters

Parameter name:

maxpop

Data type: Integer

Value: 10000000

☐ Null Value

Using the Params property

- When you enter a query, Delphi creates a *Params* array for the parameters of a dynamic SQL statement.

- *Params* is a zero-based array of *TParam* objects with an element for each parameter in the query; that is, the first parameter is *Parms*, the second *Params*, and so on.

Using the Params property

- For example, suppose a *TQuery* component named *Query2* has the following statement for its *SQL* property:

```
INSERT
INTO COUNTRY (NAME, CAPITAL,
POPULATION)
VALUES
(:Name, :Capital, :Population)
```

Using the Params property

- An application could use *Params* to specify the values of the parameters as follows:

```
Query2.Params[0].AsString := 'Lichtenstein';
Query2.Params[1].AsString := 'Vaduz';
Query2.Params[2].AsInteger := 420000;
```

Using the ParamByName method

- *ParamByName* is a function that enables an application to assign values to parameters based on their names.

- Instead of providing the ordinal location of the parameter, you must supply its name.

Using the ParamByName method

- For example, an application could use *ParamByName* could specify values for the parameters in the preceding example as follows:

```
Query2.ParamByName('Name').AsString := 'Lichtenstein';
Query2.ParamByName('Capital').AsString := 'Vaduz';
Query2.ParamByName('Population').AsInteger := 420000;
```

Using the DataSource property

- For parameters of a query not bound to values at design time, Delphi will check the query’s *DataSource* property.

- This property specifies the name of a *TdataSource* component.

Using the DataSource property

- The LINKQRY sample application illustrates the use of the *DataSource* property to link a query in a master-detail form. The form contains a *TQuery* component (named Orders) with the following in its *SQL* property:

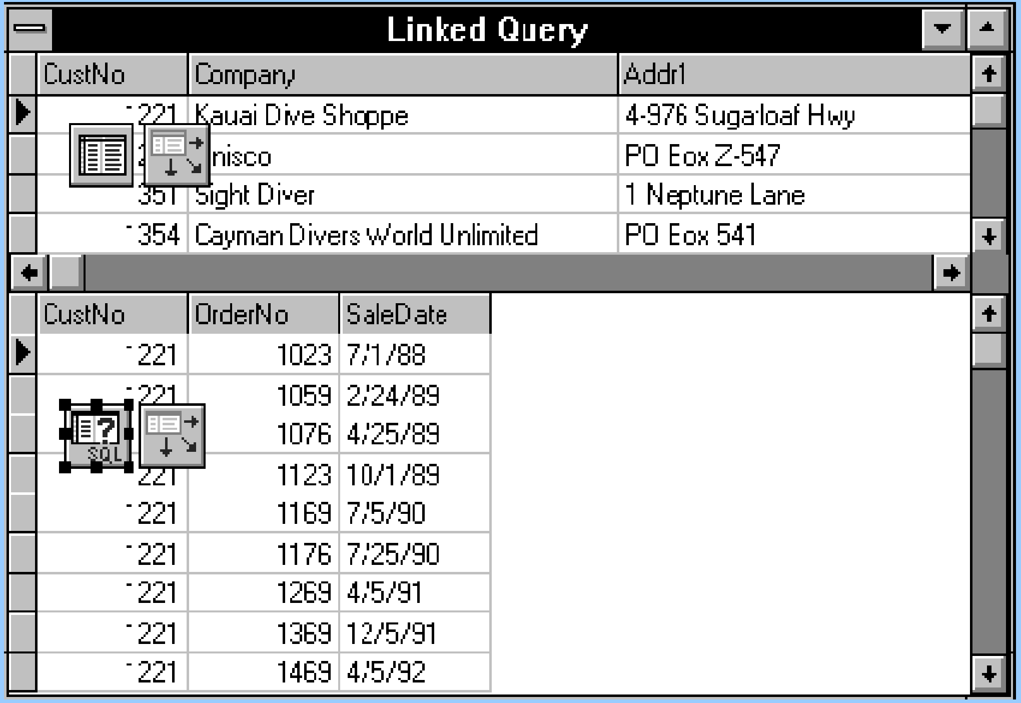
```
SELECT Orders.CustNo, Orders.OrderNo,
Orders.SaleDate
FROM Orders
WHERE Orders.CustNo = :CustNo
```

Using the DataSource property

- As illustrated below, the form also contains:
  - *ATDataSource* named OrdersSource, linked to Orders by its *DataSet* property.
  - *ATTable* component (named Cust).
  - *ATDataSource* named CustSource linked to Cust.
  - Two data grids; one linked to CustSource and the other to OrdersSource

Using the DataSource property

Form with linked queries



Linked Query

CustNo	Company	Addr1
221	Kauai Dive Shoppe	4-976 Sugarloaf Hwy
[illegible]	nisco	PO Eox Z-547
351	Sight Diver	1 Neptune Lane
354	Cayman Divers World Unlimited	PO Eox 541

CustNo	OrderNo	SaleDate
221	1023	7/1/88
221	1059	2/24/89
[illegible]	1076	4/25/89
221	1123	10/1/89
221	1169	7/5/90
221	1176	7/25/90
221	1269	4/5/91
221	1369	12/5/91
221	1469	4/5/92

Dynamic SQL example

```
SELECT * FROM country WHERE name LIKE :CountryName
```

- Prepare the query in the *OnCreate* event of the form:

```
procedure TForm1.FormCreate(Sender:
TObject);
begin
Query1.Prepare;
end;
```

Dynamic SQL example

- Provide parameters in response to some event. In this example, double-click on Button1 to edit the *OnClick* event and use the contents of Edit1.Text as a substitution parameter:

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
Query1.Close;
Query1.Params[0].AsString := Edit1.Text;
Query1.Open;
end;
```