

Microsoft Access XP

Manual - Advanced Level



SAMPLE

© 1995-2010 Cheltenham Courseware Pty. Ltd.

All trademarks acknowledged. E&OE.

No part of this document may be copied without written permission from Cheltenham Courseware unless produced under the terms of a courseware site license agreement with Cheltenham Courseware.

All reasonable precautions have been taken in the preparation of this document, including both technical and non-technical proofing. Cheltenham Courseware and all staff assume no responsibility for any errors or omissions. No warranties are made, expressed or implied with regard to these notes. Cheltenham Courseware shall not be responsible for any direct, incidental or consequential damages arising from the use of any material contained in this document. If you find any errors in these training modules, please inform Cheltenham Courseware. Whilst every effort is made to eradicate typing or technical mistakes, we apologise for any errors you may detect. All courses are updated on a regular basis, so your feedback is both valued by us and will help us to maintain the highest possible standards.

Sample versions of courseware from Cheltenham Courseware

(Normally supplied in Adobe Acrobat format): If the version of courseware that you are viewing is marked as NOT FOR TRAINING, SAMPLE, or similar, then it cannot be used as part of a training course, and is made available purely for content and style review. This is to give you the opportunity to preview our courseware, prior to making a purchasing decision. Sample versions may not be re-sold to a third party.

For current license information

This document may only be used under the terms of the license agreement from Cheltenham Courseware. Cheltenham Courseware reserves the right to alter the licensing conditions at any time, without prior notice. Please see the site license agreement available at: www.cheltenhamcourseware.com.au/agreement

Contact Information

Australia / Asia Pacific / Europe (ex. UK) / Rest of the World

Email: info@cheltenhamcourseware.com.au
Web: www.cheltenhamcourseware.com.au

USA / Canada

Email: info@cheltenhamcourseware.com
Web: www.cheltenhamcourseware.com

UK

Email: info@cctglobal.com
Web: www.cctglobal.com



SAMPLE

WORKING WITH THE NORTHWIND SAMPLE DATABASE	7
INSTALLING THE NORTHWIND DATABASE	7
<i>About Northwind</i>	7
<i>Checking for Northwind</i>	7
<i>Installing Northwind</i>	7
REVIEW QUESTIONS.....	8
INTRODUCING STRUCTURED QUERY LANGUAGE	9
WRITING SQL COMMANDS.....	10
<i>Structured Query Language (SQL) Bases</i>	10
<i>Understanding SQL Statements</i>	10
<i>Understanding SQL Conventions</i>	10
<i>Understanding SQL Syntax</i>	10
WRITING SQL QUERIES.....	11
<i>Using SELECT Statement</i>	11
<i>Selecting All Data</i>	11
<i>Selecting Specific Data</i>	12
<i>Selecting Conditional Data</i>	13
<i>Using ORDER BY Statement</i>	14
<i>Sorting Data</i>	14
<i>Using Aggregate Functions</i>	15
<i>Calculating Data</i>	15
<i>Using GROUP BY Statement</i>	16
<i>Grouping Data</i>	16
WRITING SQL SUBQUERIES.....	16
<i>Creating a Subquery with Equality</i>	16
<i>Creating a Subquery with an Aggregate Function</i>	17
<i>Using ANY and ALL Statements</i>	17
WRITING SQL JOINS.....	18
<i>Understanding SQL Joins</i>	18
<i>Creating a Simple Join</i>	18
<i>Sorting a Join</i>	19
<i>Creating Outer Joins</i>	19
<i>Creating Left Outer Joins</i>	20
<i>Creating Right Outer Joins</i>	20
<i>Creating Full Outer Joins</i>	21
INSERTING DATA.....	21
<i>Using INSERT Statement</i>	21
<i>Understanding INSERT Statement Rules</i>	22
<i>Inserting Data using VALUES</i>	22
<i>Inserting Data using Defaults</i>	22
<i>Inserting Data using SELECT</i>	22
UPDATING DATA.....	23
<i>Using UPDATE Statement</i>	23
<i>Updating All Rows</i>	23
<i>Updating Specific Rows</i>	23
<i>Updating Multiple Columns</i>	23
DELETING DATA.....	24
<i>Using DELETE FROM Statement</i>	24
<i>Deleting Specific Rows</i>	24
<i>Deleting All Rows</i>	24
WRITING SQL QUERIES IN ACCESS 2002.....	24
<i>Viewing SQL in Access 2002</i>	24

<i>Understanding SQL in Access 2002</i>	26
<i>Writing SQL Specific Queries</i>	27
<i>Finding Help on SQL Queries</i>	28
REVIEW QUESTIONS.....	29
WORKING WITH MACROS	31
AUTOMATING TASKS.....	31
<i>Understanding Macros</i>	31
INTRODUCING THE MACRO WINDOW TOOLBAR.....	32
<i>Using the Macro Window Toolbar</i>	32
CREATING MACROS.....	34
<i>Creating New Macros</i>	34
TESTING AND DEBUGGING MACROS.....	38
<i>Running a Macro</i>	38
<i>Stepping Through a Macro</i>	39
MODIFYING MACROS.....	40
<i>Modifying a Macro</i>	41
CONDITIONAL PROGRAMMING IN MACROS.....	41
<i>Adding Conditions to Macros</i>	41
<i>Using the Expression Builder to create Conditions</i>	42
<i>Running Macros with Conditions</i>	44
ADDING MACROS TO FORMS.....	45
<i>Attaching Macros to a Form</i>	45
ADDING MACROS TO REPORTS.....	47
<i>Attaching Macros to a Report</i>	47
FILTERING DATA.....	49
<i>Filtering Records</i>	49
DOCUMENTING MACROS.....	53
<i>Commenting Macros</i>	53
<i>Printing Macro Definitions</i>	54
<i>Viewing Macro Definitions</i>	54
REVIEW QUESTIONS.....	55
PROGRAMMING ACCESS USING VISUAL BASIC	56
USING MACROS VERSUS VISUAL BASIC.....	56
<i>Using Visual Basic Modules</i>	56
<i>Converting Macros to Visual Basic Code</i>	56
UNDERSTANDING VISUAL BASIC CONCEPTS.....	61
<i>Understanding Modules</i>	61
<i>Creating Modules</i>	61
<i>Understanding Module Declarations</i>	63
<i>Understanding Procedures</i>	63
<i>Using Naming Rules</i>	64
<i>Declaring Variables</i>	64
<i>Setting Variable Scope</i>	64
<i>Declaring Constants</i>	65
<i>Using Methods</i>	65
<i>Using Arguments</i>	65
USING THE VISUAL BASIC EDITOR WINDOW.....	66
<i>Customizing the Visual Basic Editor Window</i>	67
<i>Setting the Visual Basic Editor Options</i>	68
GETTING HELP WITH VISUAL BASIC.....	70
<i>Using Microsoft Visual Basic Help</i>	70
<i>Getting Visual Basic Syntax Help</i>	71
REVIEW QUESTIONS.....	73

USING DATA ACCESS PAGES.....	74
CREATING DATA ACCESS PAGES	74
<i>Using Data Access Pages.....</i>	74
<i>Creating pages using the AutoPage</i>	74
<i>Creating pages using the Page Wizard.....</i>	78
<i>Creating pages using an Existing HTML Document.....</i>	85
<i>Creating pages using the Design View.....</i>	86
MODIFYING DATA ACCESS PAGES.....	88
<i>Choosing a Theme.....</i>	88
<i>Adding Text.....</i>	90
<i>Adding Controls.....</i>	91
<i>Formatting Text and Labels.....</i>	92
<i>Aligning Content.....</i>	93
<i>Sizing Content.....</i>	93
<i>Adding Images.....</i>	94
<i>Adding Backgrounds</i>	94
SORTING DATA IN DATA ACCESS PAGES	95
<i>Sorting Data.....</i>	95
SUMMARIZING DATA IN DATA ACCESS PAGES.....	96
<i>Summarizing Data.....</i>	96
EDITING DATA IN DATA ACCESS PAGES.....	97
<i>Adding a New Record.....</i>	97
<i>Deleting a Record.....</i>	98
PROTECTING DATA IN DATA ACCESS PAGES	98
<i>Customizing Navigation Bar.....</i>	98
<i>Protecting Fields.....</i>	99
REVIEW QUESTIONS.....	100
CONVERTING ACCESS DATA BASES	101
CONVERTING DATABASES TO ACCESS 2002.....	101
<i>Converting Databases.....</i>	101
<i>Converting Database Objects.....</i>	102
<i>Enabling Databases.....</i>	103
<i>Sharing Databases across various Access versions.....</i>	104
CONVERTING FROM ACCESS 2002 TO ACCESS 97.....	106
<i>Converting an Access 2002 database to an Access 97 database.....</i>	106
CONVERTING FROM ACCESS 2002 TO ACCESS 2000.....	107
<i>Converting an Access 2002 database to an Access 2000 database.....</i>	107
REVIEW QUESTIONS.....	108
CUSTOMIZING ACCESS 2002.....	109
CONFIGURING ACCESS 2002 OPTIONS.....	109
<i>Setting Access 2002 Options.....</i>	109
<i>Setting View Options.....</i>	111
<i>Setting General Options.....</i>	112
<i>Setting Edit/Find Options.....</i>	114
<i>Setting Keyboard Options.....</i>	115
<i>Setting Datasheet Options.....</i>	116
<i>Setting Forms/Reports Options.....</i>	117
<i>Setting Pages Options.....</i>	118
<i>Setting Advanced Options.....</i>	119
<i>Setting International Options.....</i>	121
<i>Setting Spelling Options.....</i>	122
<i>Setting Tables/Queries Options.....</i>	123

CONFIGURING AUTOCORRECT OPTIONS.....	124
<i>Setting AutoCorrect Options</i>	124
<i>Excluding AutoCorrect Rules</i>	125
<i>Removing AutoCorrect Rules</i>	125
<i>Adding Replace AutoCorrect Rules</i>	126
<i>Removing Replace AutoCorrect Rules</i>	126
CUSTOMIZING MENUS AND TOOLBARS.....	126
<i>Opening Customize Dialog Box</i>	126
<i>Customizing Toolbars</i>	127
<i>Creating Custom Toolbars</i>	127
<i>Removing Toolbars</i>	130
<i>Customizing Commands</i>	131
<i>Adding Buttons to Toolbars</i>	132
<i>Removing Buttons from Toolbars</i>	133
<i>Customizing Options</i>	133
REVIEW QUESTIONS.....	134
ANALYZING ACCESS DATABASES.....	136
OPTIMIZING DATABASES WITH TABLE ANALYZER.....	136
<i>Using the Table Analyzer</i>	136
<i>Renaming Tables</i>	140
<i>Adding Key Fields</i>	141
OPTIMIZING DATABASES WITH PERFORMANCE ANALYZER.....	143
<i>Using the Performance Analyzer</i>	143
DOCUMENTING DATABASES WITH DOCUMENTER ANALYZER.....	146
<i>Using the Documenter Analyzer</i>	146
<i>Printing Documentation</i>	148
<i>Exporting Documentation</i>	149
REVIEW QUESTIONS.....	150

SAMPLE

Working with the Northwind Sample Database

When you have completed this learning module you will have seen how to:

- Check for Northwind
- Install Northwind

Installing the Northwind Database

About Northwind

- **Northwind** is a sample database that is supplied with Access 2002. It contains product and sales data for the fictitious company **Northwind Traders**.

It is recommended that Northwind be installed in order to follow the lessons in this manual.

Checking for Northwind

- Click the **Search** button on the **database** toolbar:



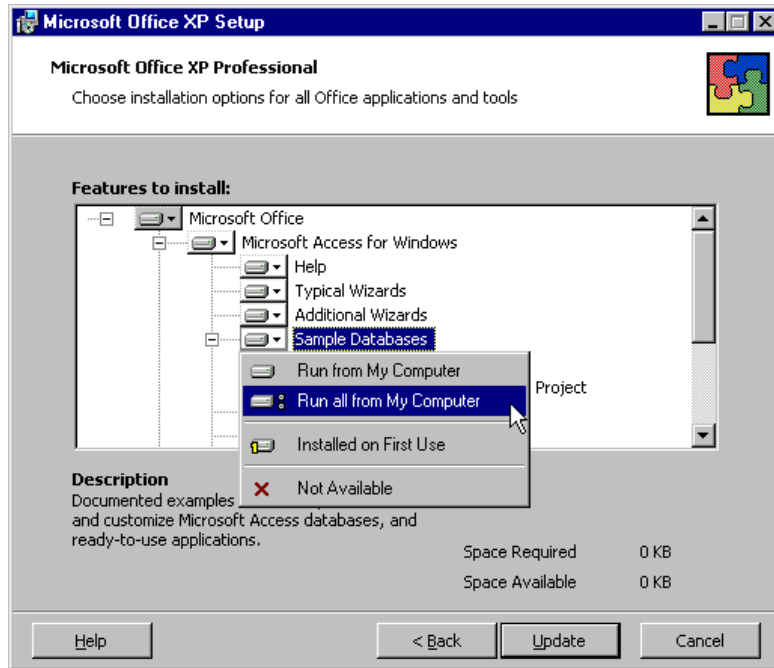
- In the Search text box, type Northwind.
- Click **Search**.
- If **Northwind.mdb** appears in the **Search Results**, click on the file to open it and proceed to the next chapter.
- If **Northwind** is not found in the **Search Results** it will need to be installed.

Installing Northwind

- Insert the Microsoft Office XP CD-ROM in the CD-ROM drive.
- Double-click **Setup**.

Note: If your computer has autorun enabled, you will be presented directly with the Microsoft Office XP setup screen.

- Select Add or Remove Features.
- Click **Next**.
- Under Features to Install, select the Microsoft Access for Windows subtree.
- Under Sample Databases, select Run all from My Computer:



- Click Update.
- Once the installer completes, click **OK** to finish.

Note: Once the **Northwind** database is installed, search for it again and open it.

Review Questions

How would you:

- Check for Northwind?
- Install Northwind?

SAMPLE

Introducing Structured Query Language

When you have completed this learning module you will have seen how to:

- Understand SQL Statements
- Understand SQL Conventions
- Understand SQL Syntax
- Use SELECT Statement
- Select All Data
- Select Specific Data
- Select Conditional Data
- Use ORDER BY Statement
- Sort Data
- Use Aggregate Functions
- Calculate Data
- Use GROUP BY Statement
- Group Data
- Create a Subquery with Equality
- Create a Subquery with an Aggregate Function
- Use ANY and ALL Statements
- Understand SQL Joins
- Create a Simple Join
- Sort a Join
- Create Outer Join
- Create Left Outer Join
- Create Right Outer Join
- Create Full Outer Join
- Use INSERT Statement
- Understand INSERT Statement Rules
- Insert Data using VALUES
- Insert Data using Defaults
- Insert Data using SELECT
- Use UPDATE Statement
- Update All Rows
- Update Specific Rows
- Update Multiple Columns
- Use DELETE FROM Statement
- Delete Specific Rows
- Delete All Rows
- View SQL in Access 2002
- Understand SQL in Access 2002
- Write SQL Specific Queries
- Find Help on SQL Queries

Writing SQL Commands

Structured Query Language (SQL) Bases

- Structured Query Language is called **SQL** for short.
- SQL consists of standard English words.
- Like all programming languages, SQL has specific conventions and grammatical syntax.
- SQL can be used by a range of users including Database Administrators, management personnel, application programmers and other types of end users.

Understanding SQL Statements

- SQL statement consists of **reserved words** and **user-defined words**:

Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.

User-defined words are created by the user and represent names of various database objects such as relations, columns, views, etc.

Understanding SQL Conventions

- SQL is more readable if indentation and lineation are used:
- Each clause in a statement should begin on a new line.
- Start of a clause should line up with start of other clauses.
- If clause has several parts, they should each appear on a separate line and be indented under the start of clause.
- Upper case letters represent SQL **reserved** words.
- Lower case letters represent **user-defined** words.
- A vertical bar (|) indicates a **choice** among alternatives.
- Curly braces ({ }) indicate a **required element**.
- Square brackets ([]) indicate an **optional element**.
- An ellipsis (...) indicates **optional repetition** (0 or more times).

Understanding SQL Syntax

- The main elements in SQL are the statements and clauses that makeup statements.
- SELECT command indicates which fields are included in the query result.
- FROM clause indicated what table(s) the fields are retrieved from.
- The following is the syntax of the SELECT statement:

```
SELECT      { * | [column_name] [, ...] }
FROM        table_name
```

Writing SQL Queries

Using SELECT Statement

- Data is selected using the **SELECT** statement.
- Order of the clauses cannot be changed.

```
SELECT      [predicate] { * | [column_name] [, ...] }
FROM        table_name
[WHERE      condition]
[GROUP BY  column_name]
[HAVING    group_condition]
[ORDER BY  column_name] ;
```

- **SELECT** command specifies which columns are to appear in the output.
- **FROM** clause specifies table or tables to be used.
- **WHERE** clause filters rows subject to some conditions.
- **GROUP BY** clause forms groups of rows with same column value.
- **HAVING** clause specifies conditions the grouped records must meet to be displayed.
- **ORDER BY** clause specifies the order of the output.
- **Semicolon ;** ends the SQL statement.
- **Predicate** is optional, and it restricts the number of results returned.
- There are four predicates:

ALL	includes all records.
DISTINCT	omits duplicate data in selected fields.
DISTINCTROW	omits data based on entire duplicate records.
TOPn [PERCENT]	limits the records to a number or percentage of records.

- Only SELECT and FROM are mandatory.
- Use * as an abbreviation for 'all columns'.

Selecting All Data

- Example 1: Retrieve All Columns, All Rows
(list full details of all staff)

```
SELECT  sno, fname, lname, position, salary
FROM    staff;
```

- Example 2: Retrieve All data
(list full details of all staff)

```
SELECT *
FROM staff;
```

The result table in either case is:

sno	fname	lname	position	bono	salary
SL21	John	White	Manager	B5	30000.00
SG37	Ann	Beech	Snr Asst	B3	12000.00
SG14	David	Ford	Deputy	B3	18000.00
SA9	Mary	How e	Assistant	B7	9000.00
SG5	Susan	Brand	Manager	B3	24000.00
SL41	Julie	Lee	Assistant	B5	9000.00

Selecting Specific Data

- Example 3: Retrieve Specific Columns, All Rows
(produce a list of salaries for all staff, showing only the staff number, sno, the first and last names, and the salary details).

```
SELECT sno, fname, lname, salary
FROM staff;
```

The result table is:

sno	fname	lname	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	How e	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

- Example 4: Calculated Fields
(produce a list of monthly salaries for all staff, showing the staff number, the first and last names, and the salary details).

```
SELECT sno, fname, lname, salary/12
FROM staff;
```

The result table is:

sno	fname	lname	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	How e	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

- Example 5: Named Calculated Fields
In the example above, the 4th column is labelled **col4**. To name this column use **AS** clause:

```
SELECT sno, fname, lname, salary/12 AS monthly_salary
FROM staff;
```

The result table is:

sno	fname	lname	monthly_salary
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

Selecting Conditional Data

- Example 6: Comparison Search Condition
(list all staff with a salary greater than 10000.00)

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > 10000;
```

The result table is:

sno	fname	lname	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Snr Asst	12000.00
SG14	David	Ford	Deputy	18000.00
SG5	Susan	Brand	Manager	24000.00

- Example 7: Compound Comparison Search Condition (AND / OR)
(list all staff with the position of Manager or Assistant)

```
SELECT sno, fname, lname, position
FROM staff
WHERE position = 'Manager' OR position = 'Assistant';
```

The result table is:

sno	fname	lname	position
SL21	John	White	Manager
SA9	Mary	Howe	Assistant
SG5	Susan	Brand	Manager
SL41	Julie	Lee	Assistant

- Example 8: Pattern Match Search Condition (LIKE / NOT LIKE)
(list all staff with any assistant position, looking for string 'Ass' in their position)

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE position LIKE '%Ass%';
```

The result table is:

sno	fname	lname	position	salary
SG37	Ann	Beech	Snr Asst	12000.00
SA9	Mary	How e	Assistant	9000.00
SL41	Julie	Lee	Assistant	9000.00

- **Note:** SQL has two special pattern matching symbols:
 % percent represents any sequence of zero or more characters.
 _ underscore character represents any single character.

Using ORDER BY Statement

- Data is sorted using the **ORDER BY** statement.
- Data is sorted in Ascending order by default.
- To sort data in Descending order, use **DESC** clause.

Sorting Data

- Example 9: Single Column Ordering
 (list salaries for all staff, arranged in descending order of salary).

```
SELECT      sno, fname, lname, salary
FROM        staff
ORDER BY    salary DESC;
```

The result table is:

sno	fname	lname	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	How e	9000.00
SL41	Julie	Lee	9000.00

- Example 10: Multiple Column Ordering
 (list of all staff, arranged in ascending order of last name and position).

```
SELECT      sno, fname, lname, position
FROM        staff
ORDER BY    lname, position;
```

The result table is:

sno	fname	lname	position
SG37	Ann	Beech	Snr Asst
SG5	Susan	Brand	Manager
SG14	David	Ford	Deputy
SA9	Mary	How e	Assistant
SL41	Julie	Lee	Assistant
SL21	John	White	Manager

Using Aggregate Functions

There are five aggregate functions:

- **COUNT** returns the number of values in a specified column.
 - **SUM** returns the sum of the values in a specified column.
 - **AVG** returns the average of the values in a specified column.
 - **MIN** returns the smallest value in a specified column.
 - **MAX** returns the largest value in a specified column.
- Each function operates on a single column of a table and return single value.
 - COUNT, MIN and MAX apply to numeric and non-numeric fields.
 - SUM and AVG apply to numeric fields only.
 - COUNT(*) is a special use of COUNT that counts all rows of a table.

Calculating Data

- Example 11: Use of COUNT(*)
(find the total number of staff that have salary of 9000).

```
SELECT COUNT(*) AS count
FROM staff
WHERE salary = 9000;
```

The result table is:

count
2

- Example 12: Use of COUNT and SUM
(find the total number of Managers and the sum of their salaries).

```
SELECT COUNT(sno) AS count, SUM(salary) AS sum
FROM staff
WHERE position = 'Manager';
```

The result table is:

count	sum
2	54000.00

- Example 13: Use of MIN, MAX, AVG
(find the minimum, maximum and average staff salary).

```
SELECT MIN(salary) AS min, MAX(salary) AS max, AVG(salary) AS avg
FROM staff;
```

The result table is:

min	max	avg
9000.00	30000.00	17000.00

Using GROUP BY Statement

- Data is grouped using the **GROUP BY** statement.
- Each item in SELECT list must be **single-valued per group**.
- All column names in SELECT list must appear in GROUP BY clause, unless the name is used only in an aggregate function.

Grouping Data

- Example 14: Use of GROUP BY (find number of staff in each branch and their total salaries).

```
SELECT      bno, COUNT(sno), AS count, SUM(salary) AS sum
FROM        staff
GROUP BY    bno
ORDER BY    bno;
```

The result table is:

bno	count	sum
B3	3	54000.00
B5	2	39000.00
B7	1	9000.00

Writing SQL Subqueries

- Some SQL statements can have a SELECT embedded within them.
- A subselect can be used in the WHERE clause of an outer SELECT, where it is called a **Subquery** or **Nested query**.

Creating a Subquery with Equality

- Example 15: Use of Subquery with Equality (list the staff who work in the branch at '163 Main St').

```
SELECT      sno, fname, lname, position
FROM        staff
WHERE       bno =
            (SELECT bno
             FROM branch
             WHERE street = '163 Main St');
```

Note: Inner SELECT finds branch number corresponding to branch at '163 Main St' which is ('B3'). Outer SELECT then retrieves details of all staff who work at this branch. The outer SELECT then becomes:

```
SELECT      sno, fname, lname, position
FROM        staff
```

```
WHERE bno = 'B3';
```

The result table is:

sno	fname	lname	position
SG37	Ann	Beech	Snr Asst
SG14	David	Ford	Deputy
SL21	John	White	Manager

Creating a Subquery with an Aggregate Function

- Example 16: Use of Subquery with Aggregate Function (list all staff whose salary is greater than the average salary).

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary >
      (SELECT AVG(salary)
       FROM staff);
```

- **Note:** You cannot write 'WHERE salary > AVG(salary)'. Instead, you can use Subquery to find the average salary (17000), and then use outer SELECT to find those staff members with a salary greater than this:

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > 17000;
```

The result table is:

sno	fname	lname	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Deputy	18000.00
SG5	Susan	Brand	Manager	24000.00

Using ANY and ALL Statements

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- If Subquery is preceded by ALL, the condition will only be true if it is satisfied by **all** values produced by the Subquery.
- If Subquery is preceded by ANY, the condition will be true if it is satisfied by **any** values produced by the Subquery.
- Example 17: Use of ANY (find staff whose salary is larger than the salary of at least 1 member of the staff at branch B3).

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > ANY
      (SELECT salary
```

```
FROM staff
WHERE bno = 'B3');
```

- **Note:** Inner SELECT produces the set {12000, 18000, 24000} and outer SELECT lists those staff whose salaries are greater than any of the values in this set.

The result table is:

sno	fname	lname	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Deputy	18000.00
SG5	Susan	Brand	Manager	24000.00

- Example 18: Use of ALL
(find staff whose salary is larger than the salary of every member of the staff at branch B3).

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > ALL
(SELECT salary
FROM staff
WHERE bno = 'B3');
```

The result table is:

sno	fname	lname	position	salary
SL21	John	White	Manager	30000.00

Writing SQL Joins

Understanding SQL Joins

- In the previous examples, subqueries have provided results from the same table.
- If result columns come from more than one table, we must use a **join**.
- To perform a join, you must include more than one table in the FROM clause, using a comma as a separator and typically including a WHERE clause to specify join column(s).
- It is also possible to use an **alias** for a table, named in the FROM clause. An alias is separated from the table name with a space.

Creating a Simple Join

- Example 19: Use of Simple Join
(list names of all renters who have viewed a property).

```
SELECT r.mo, fname, lname, pno
```

```
FROM renter r, viewing v
WHERE r.mo = v.mo;
```

- **Note:** To obtain correct rows, include only those rows from both tables that have identical values in the **rno** columns: $r.mo = v.mo$. These two columns are the matching columns for two tables.

The result table is:

mo	fname	lname	pno
CR56	Aline	Stewart	PG36
CR56	Aline	Stewart	PA14
CR56	Aline	Stewart	PG4
CR62	Mary	Tregear	PA14
CR76	John	Kay	PG4

Sorting a Join

- Example 20: Sort a Join
(for each branch, list names of staff who manage properties).

```
SELECT s.bno, s.sno, fname, lname, pno
FROM staff s, property_for_rent p
WHERE s.sno = p.sno
ORDER BY s.bno, s.sno, pno;
```

The result table is:

bno	sno	fname	lname	pno
B3	SG14	David	Ford	PG4
B3	SG14	David	Ford	PG16
B3	SG37	Ann	Beech	PG21
B3	SG37	Ann	Beech	PG36
B5	SL41	Julie	Lee	PL94
B7	SA9	Mary	Howe	PA14

Creating Outer Joins

- With an inner join, if one row of a table is unmatched, the row is omitted from the result table.
- Outer join retains rows that do not satisfy the join condition.
- Consider the following two simplified tables:

BRANCH1

bno	bcity
B3	Glasgow
B4	Bristol
B2	London

PROPERTY_FOR_RENT1

pno	pcity
PA14	Aberdeen

PL94	London
PG4	Glasgow

- Example 21: The INNER JOIN of these two tables

```
SELECT b.*, p.*
FROM branch1 b, property_for_rent1 p
WHERE b.bcity = p.pcity;
```

OR

```
SELECT b.*, p.*
FROM branch1 b INNER JOIN property_for_rent1 p
ON b.bcity = p.pcity;
```

The result table is:

bno	bcity	pno	pcity
B3	Glasgow	PG4	Glasgow
B2	London	PL94	London

- **Note:** The result table has two rows where the cities are the same. There are no rows corresponding to the branches in Bristol and Aberdeen.

Creating Left Outer Joins

- Example 22: Left Outer Join
(list branches and properties that are in the same city along with any unmatched branches).

```
SELECT b.*, p.*
FROM branch1 b LEFT JOIN property_for_rent1 p
ON b.bcity = p.pcity;
```

The result table is:

bno	bcity	pno	pcity
B3	Glasgow	PG4	Glasgow
B4	Bristol	NULL	NULL
B2	London	PL94	London

- **Note:** Left outer join includes those rows of first (left) table that are unmatched with rows from second (right) table. Columns from second table are filled with NULLs.

Creating Right Outer Joins

- Example 23: Right Outer Join
(list branches and properties in the same city and any unmatched properties).

```
SELECT b.*, p.*
```

```
FROM branch1 b RIGHT JOIN property_for_rent1 p
ON b.bcity = p.pcity;
```

The result table is:

bno	bcity	pno	pcity
NULL	NULL	PA14	Aberdeen
B3	Glasgow	PG4	Glasgow
B2	London	PL94	London

- **Note:** Right outer join includes those rows of second (right) table that are unmatched with rows from first (left) table. Columns from first table are filled with NULLs.

Creating Full Outer Joins

- Example 24: Full Outer Join
(list branches and properties in the same city and any unmatched branches and properties).

```
SELECT b.*, p.*
FROM branch1 b FULL JOIN property_for_rent1 p
ON b.bcity = p.pcity;
```

The result table is:

bno	bcity	pno	pcity
NULL	NULL	PA14	Aberdeen
B3	Glasgow	PG4	Glasgow
B4	Bristol	NULL	NULL
B2	London	PL94	London

- **Note:** Full outer join includes those rows that are unmatched in both tables. Unmatched columns are filled with NULLs.

Inserting Data

Using INSERT Statement

- Data is inserted in the table using the INSERT statement.

```
INSERT INTO table_name [ (column_list) ]
```

```
VALUES (data_value_list);
```

- **Column_list** is optional.
- If omitted, SQL assumes a list of all columns in their original order.
- Any columns omitted must be declared as NULL when the table is created, unless DEFAULT is specified when creating the column.

Understanding INSERT Statement Rules

Data_value_list must match **column_list** as follows:

- Number of items in each list must be the same.
- Order of items must correspond directly to the position of items in two lists.
- Data type of each item in **data_value_list** must be compatible with data type of the corresponding column.

Inserting Data using VALUES

- Example 25: INSERT ... VALUES
(insert a new record into **staff** table supplying data for all columns).

```
INSERT INTO  staff
VALUES      ('SG16', 'Alan', 'Brown', 'Assistant', 'B3', 8300);
```

Inserting Data using Defaults

- Example 26: INSERT using Defaults
(insert a new record into **staff** table supplying data for all mandatory columns).

```
INSERT INTO  staff (sno, fname, lname, position, bno
VALUES      ('SG16', 'Alan', 'Brown', 'Assistant', 'B3');
```

OR

```
INSERT INTO  staff
VALUES      ('SG16', 'Alan', 'Brown', 'Assistant', 'B3', NULL);
```

Inserting Data using SELECT

- Second form of INSERT allows multiple rows to be copied from one or more tables to another.

```
INSERT INTO table_name [ (column_list) ]
SELECT ...;
```

- Example 27: INSERT ... SELECT
Assume there is a table **staff_prop_count** that contains names of staff and the number of properties they manage:
`staff_prop_count(sno, fname, lname, prop_cnt)`.
(populate **staff_prop_count** using **staff** and **property_for_rent**).

```
INSERT INTO  staff_prop_count
(SELECT s.sno, fname, lname, COUNT(*)
FROM staff s, property_for_rent p
WHERE s.sno = p.sno
GROUP BY s.sno, fname, lname);
```

Updating Data

Using UPDATE Statement

- Data is updated in the table using the UPDATE statement.

```
UPDATE table_name
SET     column_name1 = data_value1
      [, column_name2 = data_value2 ...]
[WHERE search_condition];
```

- SET clause specifies names of one or more columns that are to be updated.
- WHERE clause is optional. If omitted, named columns are updated for all rows in table. If specified, only those rows that satisfy **search_condition** are updated.
- New **data_value(s)** must be compatible with the data type for the corresponding column.

Updating All Rows

- Example 28: UPDATE All Rows (give all staff a 3% pay increase).

```
UPDATE staff
SET     salary = salary*1.03;
```

Updating Specific Rows

- Example 29: UPDATE Specific Rows (give all Managers a 5% pay increase).

```
UPDATE staff
SET     salary = salary*1.05
WHERE  position = 'Manager';
```

Note: WHERE clause finds rows that contain data for Managers. Update is applied only to these particular rows.

Updating Multiple Columns

- Example 30: UPDATE Multiple Columns (promote David Ford (sno = 'SG14') to Manager and change his salary to 18000).

```
UPDATE staff
SET     position = 'Manager', salary = 18000
WHERE  sno = 'SG14';
```

Deleting Data

Using DELETE FROM Statement

- Data is removed from the table using the **DELETE FROM statement**.

```
DELETE FROM table_name  
[WHERE search_condition];
```

- **Search_condition** is optional. If omitted, all rows are deleted from the table.
- This does not delete the table.
- If **search_condition** is specified, only those rows that satisfy the condition are deleted.

Deleting Specific Rows

- Example 31: DELETE Specific Rows
(delete all staff that hold a Manager position).

```
DELETE FROM staff  
WHERE position = 'Manager';
```

Deleting All Rows

- Example 32: DELETE All Rows
(delete all records from the **staff** table).

```
DELETE FROM staff;
```

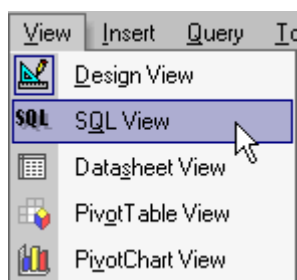
Writing SQL Queries in Access 2002

Viewing SQL in Access 2002

- SQL is the language that Access 2002 uses to program query operations.
- To view and/or edit SQL statements while creating a **Query**, switch from **Design View** to **SQL View**:

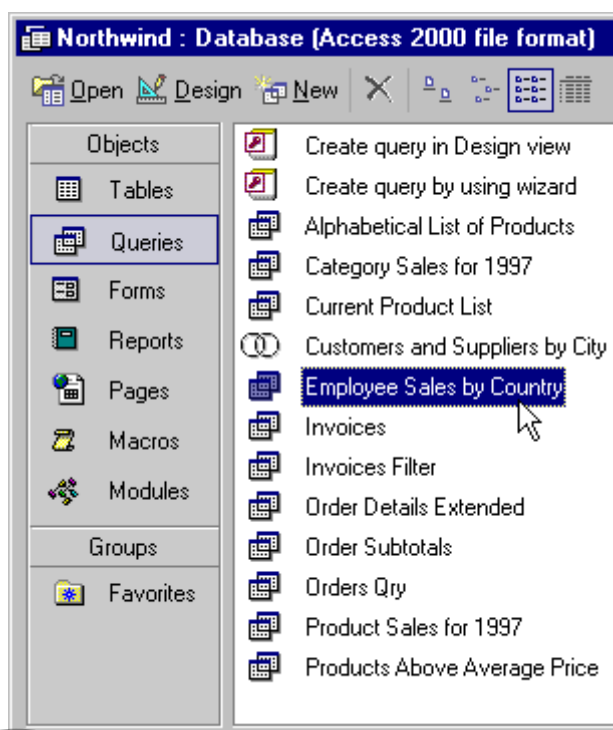
To open the SQL View:

- Open a Query in **Design View**.
- From the main menu, choose **View > SQL View**:

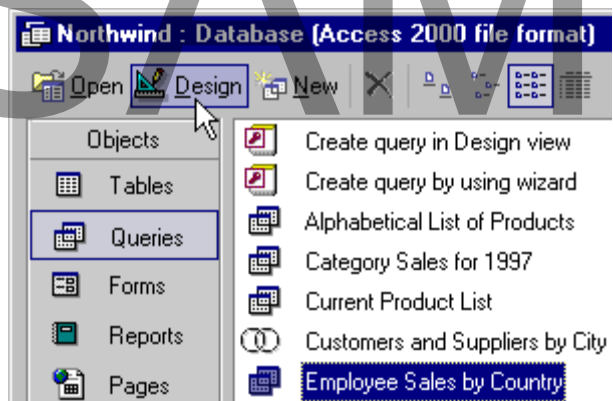


To view sample SQL statement:

- For example, open the sample **Northwind.mdb** database.
- Click on the **Queries** button under the **Objects** pane of the **Database** window.
- Select a query named **Employee Sales by Country**:

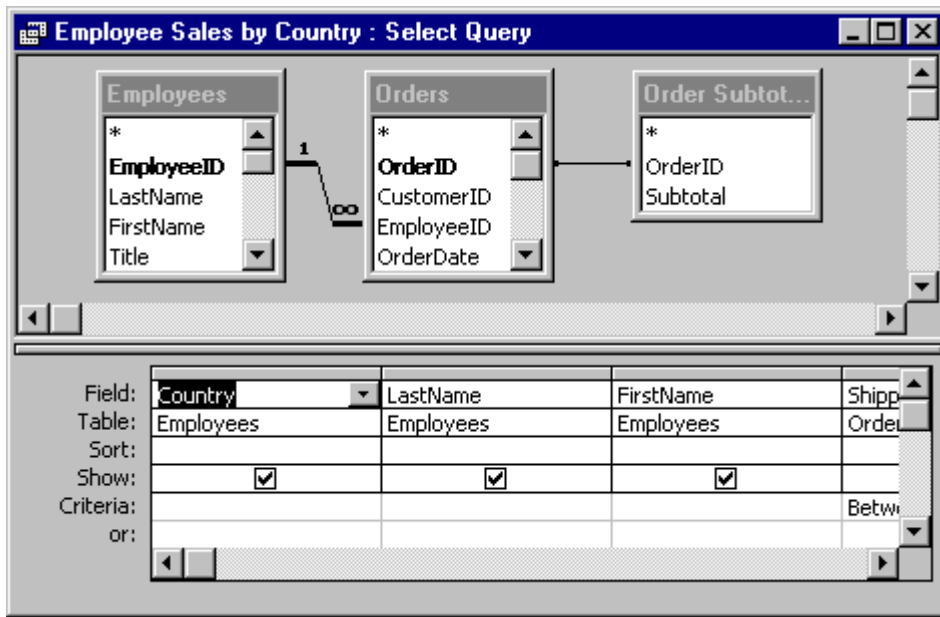


- Click on the **Design** button:

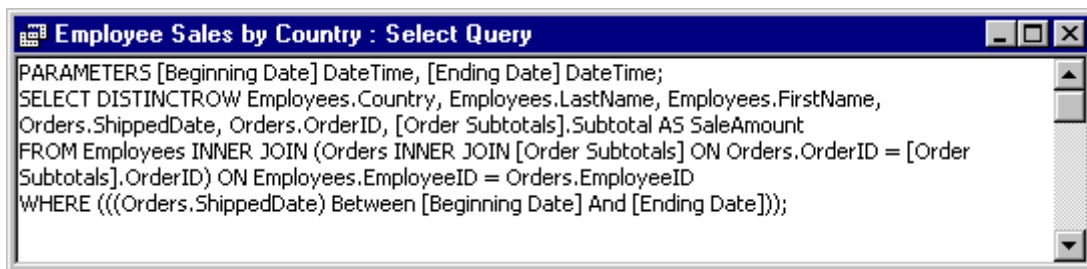


SAMPLE

- This will open the **Design View** for the selected query:



- While in **Design View**, from the main menu, choose **View > SQL View**.
- This will open the **SQL View**:



- You can edit the SQL statement directly in this window.

Understanding SQL in Access 2002

- SQL statements are little different in Access that in the standard SQL.
- Lets look at the example:

```
SELECT [First Name], [Last Name], Position
FROM [Staff Data]
WHERE Position = 'Assistant';
```

- The column names that contain spaces must be enclosed in square brackets [].
- If the column name does not contain spaces, square brackets are not needed.

Writing SQL Specific Queries

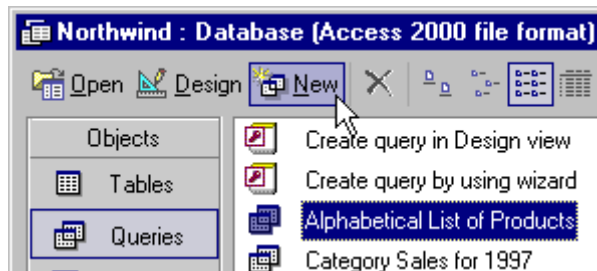
- SQL Specific queries are queries that can be created only using SQL statements.

There are three SQL Specific queries:

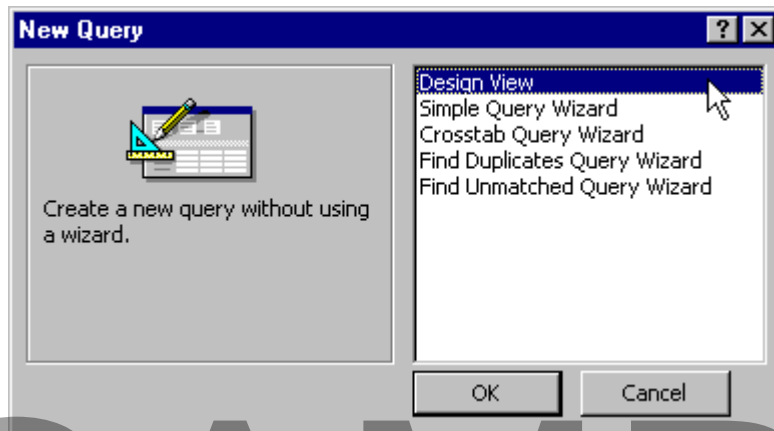
- **Union** queries that combine corresponding fields from two or more tables into one field in the query results.
- **Pass-Through** queries that send commands directly to ODBC databases.
- **Data Definition** queries that create or edit Access tables.

To write SQL Specific query:

- Click on the **New** button while in the **Queries** page of the Database window:

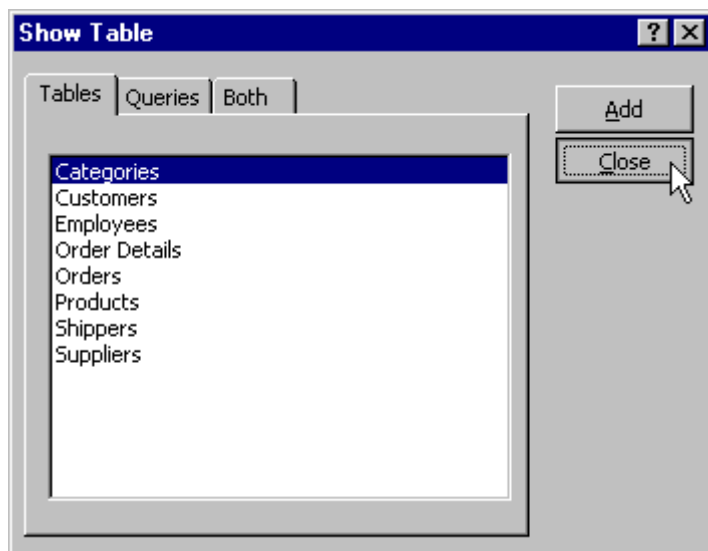


- In the **New Query** dialog box, select **Design View** and click **OK**:

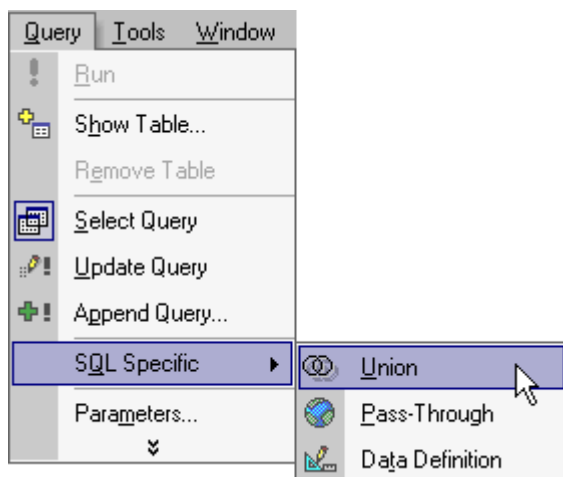


- In the **Show Table** dialog box, click **Close** without adding any table:

SAMPLE

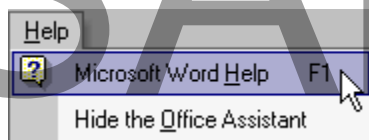


- From the main menu, choose **Query > SQL Specific** > then select the type of query you want to create, **Union**, **Pass-Through** or **Data Definition**:



Finding Help on SQL Queries

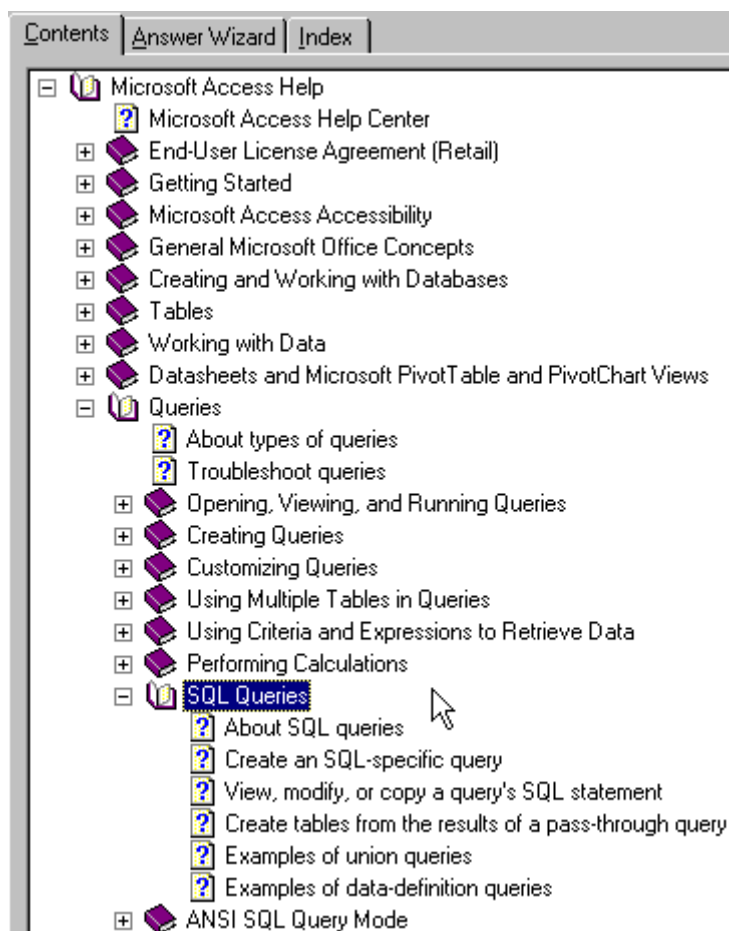
- As always, if you need more help, Access 2002 has more information on SQL queries.
- From the main menu, choose **Help > Microsoft Access Help**:



OR press the **F1** key

OR click on the **Help** [?] toolbar button.

- In the **Contents** page, navigate to **Queries > SQL Queries** to list all subjects on this issue:



Review Questions

How would you:

- Understand SQL Statements?
- Understand SQL Conventions?
- Understand SQL Syntax?
- Use SELECT Statement?
- Select All Data?
- Select Specific Data?
- Select Conditional Data?
- Use ORDER BY Statement?
- Sort Data?
- Use Aggregate Functions?
- Calculate Data?
- Use GROUP BY Statement?
- Group Data?
- Create a Subquery with Equality?
- Create a Subquery with an Aggregate Function?
- Use ANY and ALL Statements?

- Understand SQL Joins?
- Create a Simple Join?
- Sort a Join?
- Create an Outer Join?
- Create a Left Outer Join?
- Create a Right Outer Join?
- Create a Full Outer Join?
- Use INSERT Statement?
- Understand INSERT Statement Rules?
- Insert Data using VALUES?
- Insert Data using Defaults?
- Insert Data using SELECT?
- Use UPDATE Statement?
- Update All Rows?
- Update Specific Rows?
- Update Multiple Columns?
- Use DELETE FROM Statement?
- Delete Specific Rows?
- Delete All Rows?
- View SQL in Access 2002?
- Understand SQL in Access 2002?
- Write SQL Specific Queries?
- Find Help on SQL Queries?

SAMPLE

Working with Macros

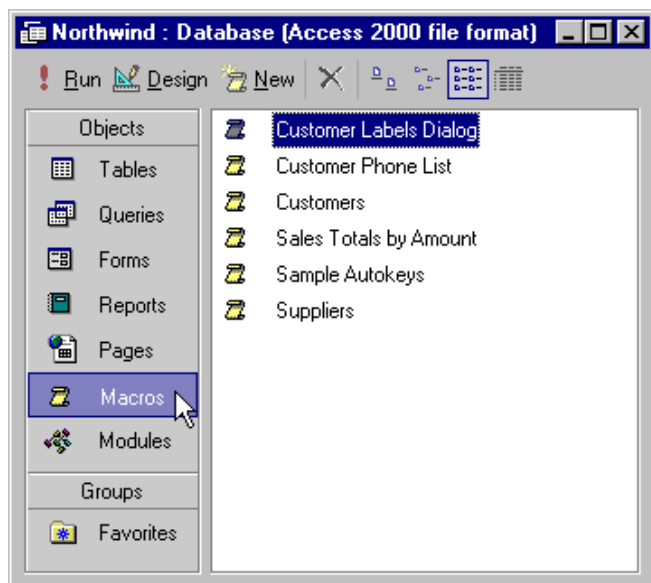
When you have completed this learning module you will have seen how to:

- Understand Macros
- Use the Macro Window Toolbar
- Create New Macros
- Run a Macro
- Step Through a Macro
- Modify a Macro
- Add Conditions to Macros
- Use the Expression Builder to Create Conditions
- Run Macros with Conditions
- Attach Macros to a Form
- Attach Macros to a Report
- Filter Records
- Comment Macros
- Print Macro Definitions
- View Macro Definitions

Automating Tasks

Understanding Macros

- Macros present a way of programming Access 2002 to perform a variety of tasks and actions.
- Macros are easy to create - you simply select from the list of predefined **actions** and their **arguments**.
- Each action performs a specific operation.
- Each action can have arguments, which specify additional information for that action.
- Macros are best for performing simple tasks, like opening and closing forms, running reports, and displaying custom toolbars.
- Macros are individual Access objects listed in the **Macros** page in the Database window:

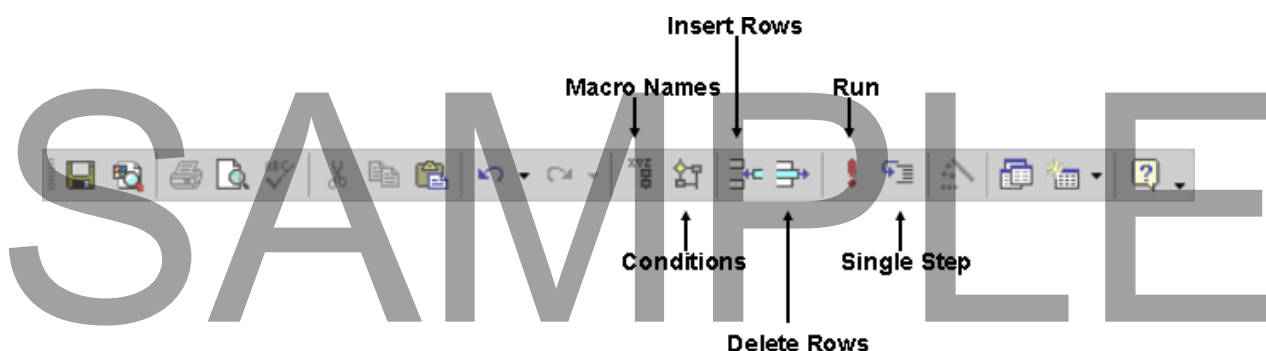


- Once macros are created, you can attach them to an **Event** property of any object in your database.
- **Note:** Access macros differ from other macros in Office XP and other applications, because it does not record the keystrokes.

There are two cases when you must use macros:

- When assigning specific actions to a key combination to be used globally, in the entire database.
- When assigning start up actions to the database, such as opening a switchboard form at start up.

Introducing the Macro Window Toolbar

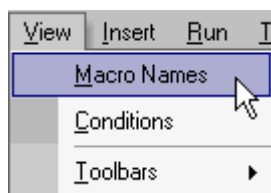


Using the Macro Window Toolbar

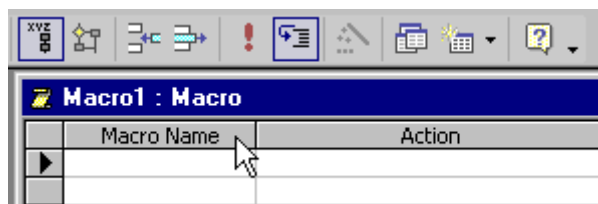
- **Macro Names** button adds the Macro Name column to the macro sheet:



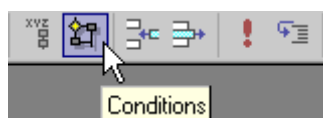
OR from the main menu, choose **View > Macro Names**:



Macro Name column is added to the macro sheet:

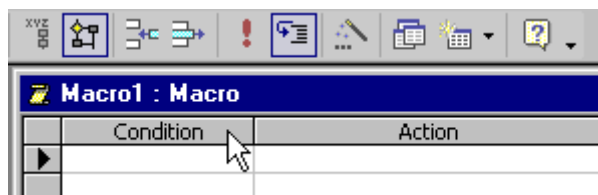


- **Conditions** button adds the Condition column to the macro sheet:



OR from the main menu, choose **View > Conditions**.

Condition column is added to the macro sheet:



- **Insert Rows** button inserts one or more blank rows in the grid above the selected row:



OR from the main menu, choose **Insert > Rows**.

- **Delete Rows** button deletes the selected row or rows:



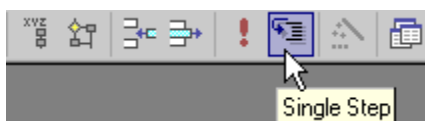
OR from the main menu, choose **Edit > Delete Rows**.

- **Run** button runs the macro:



OR from the main menu, choose **Run > Run**.

- **Single Step** button runs the macro one action at a time, displaying intermediate information:

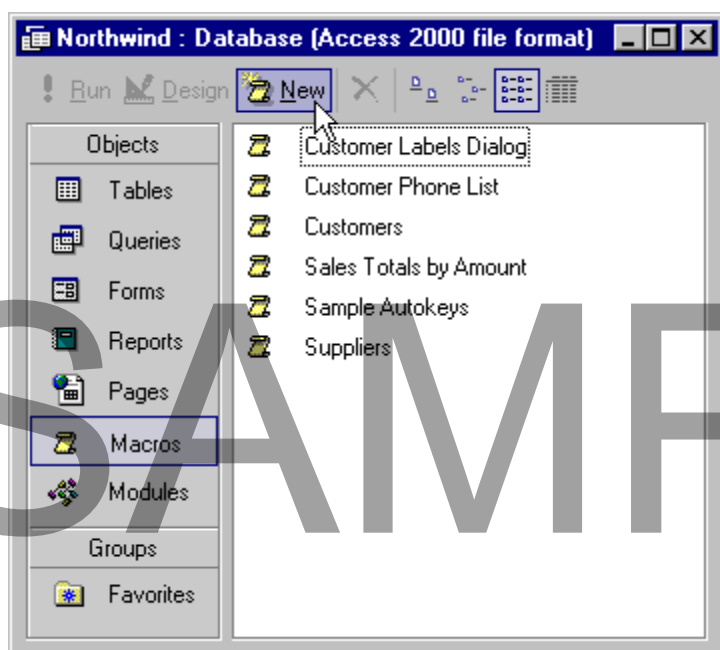


OR from the main menu, choose **Run > Single Step**.

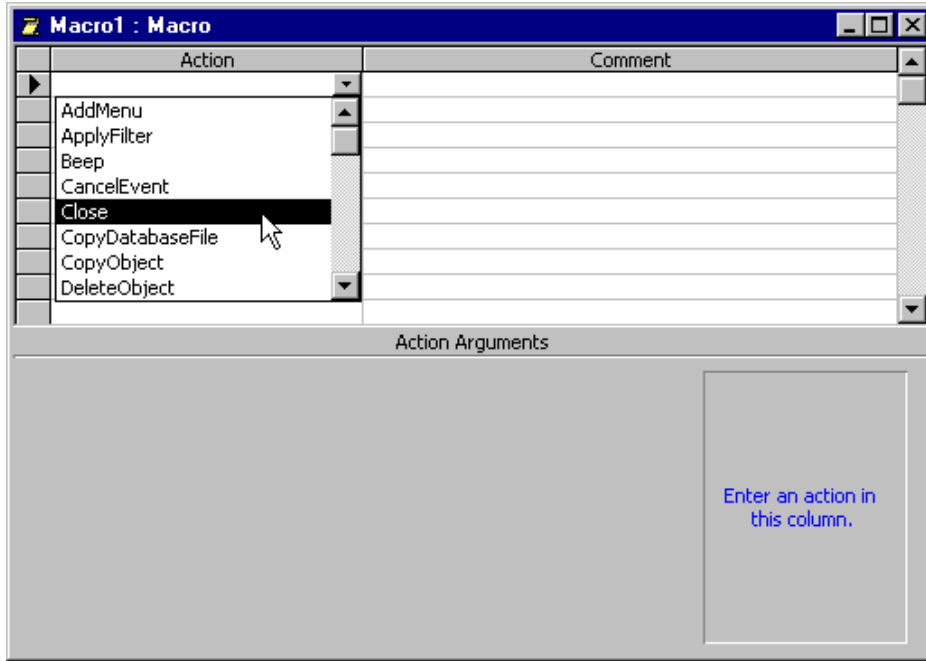
Creating Macros

Creating New Macros

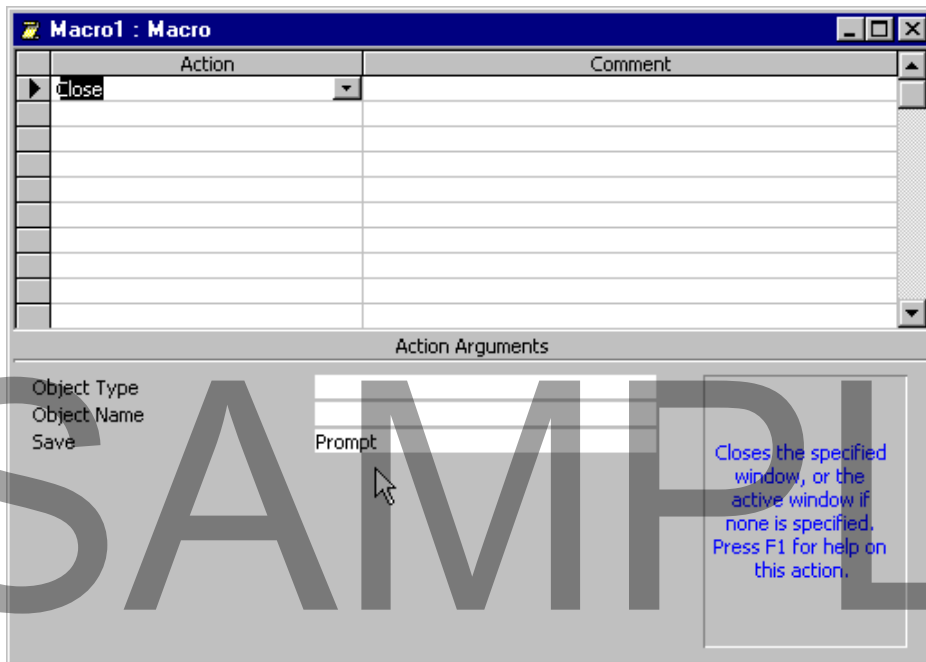
- While in the **Macros** page of the **Database** window, click on the **New** button:



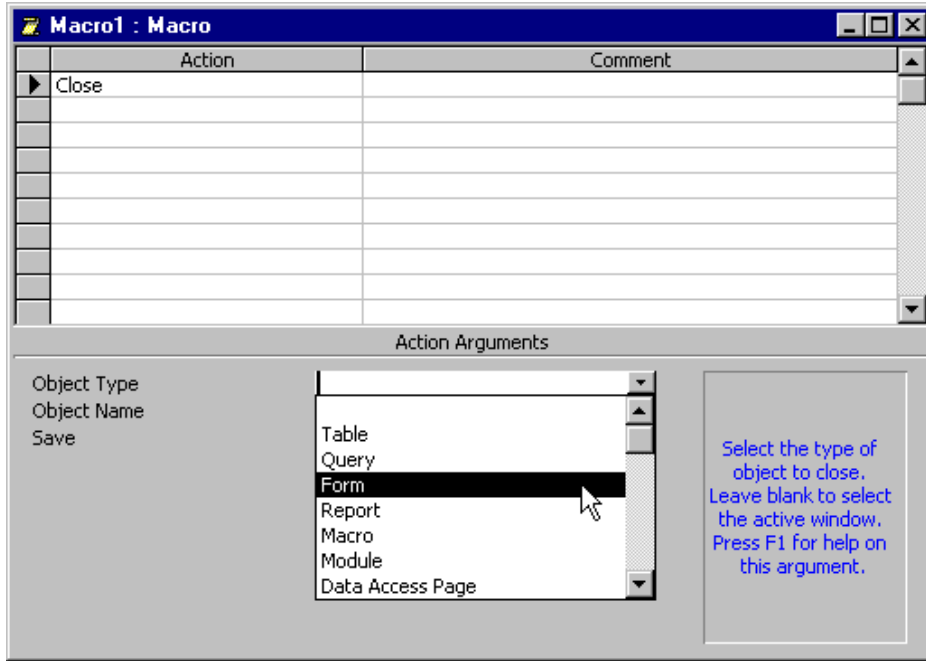
- The **Macro** window opens, displaying the blank macro sheet.
- Under the **Action** column, you will see the drop-down list which contains a list of predefined actions for you to choose from:



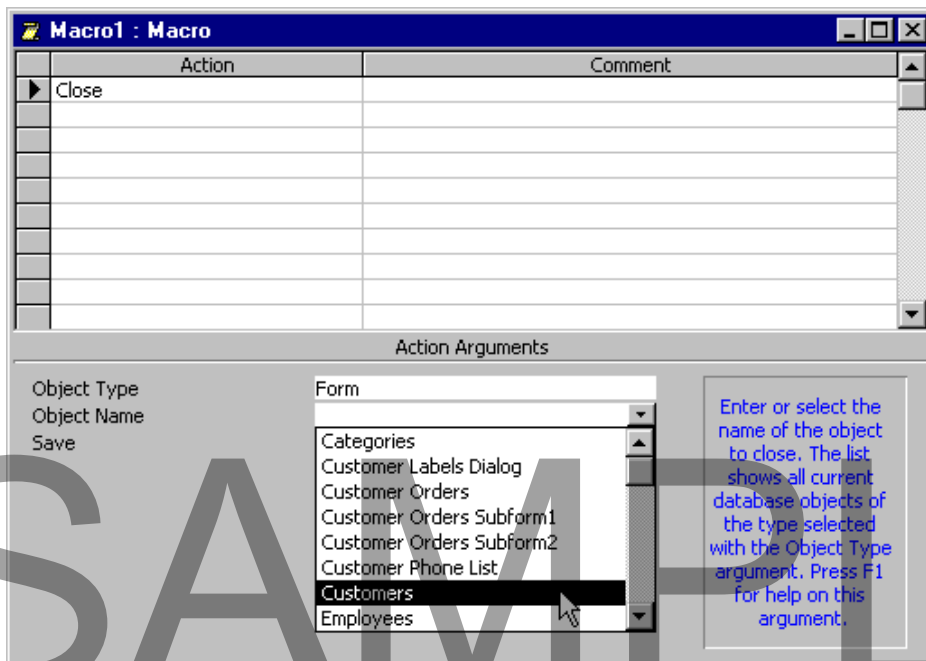
- The **Comment** column contains optional comments used for documenting macros.
- Once you select an Action, the pane on the bottom area displays the associated **Action Arguments**:



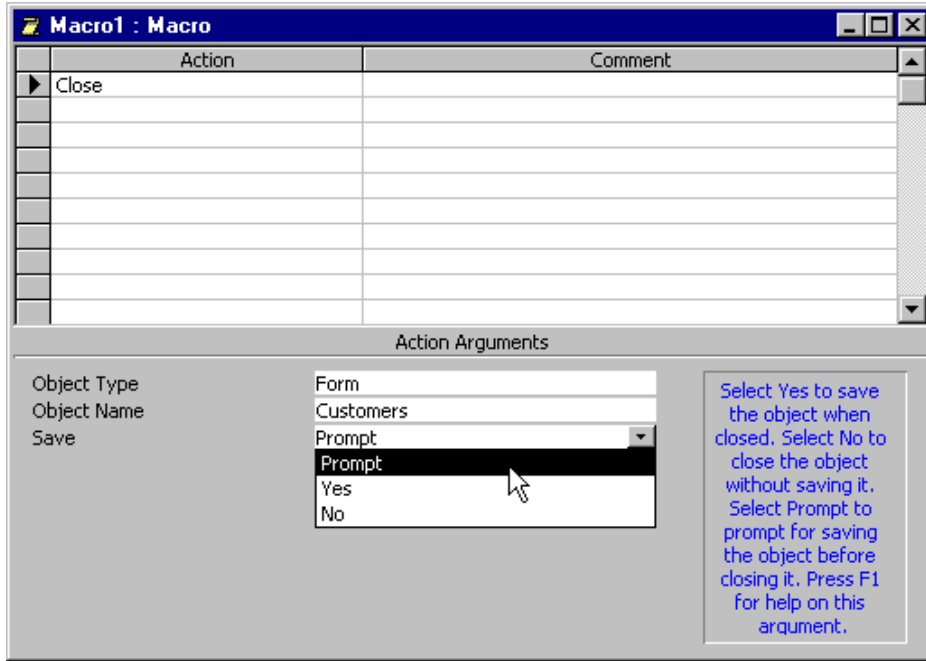
- Depending on the selected action, some arguments are required and some are optional.
- For example, for the **Close** action, you can select an **Object Type**, such as **Form**:



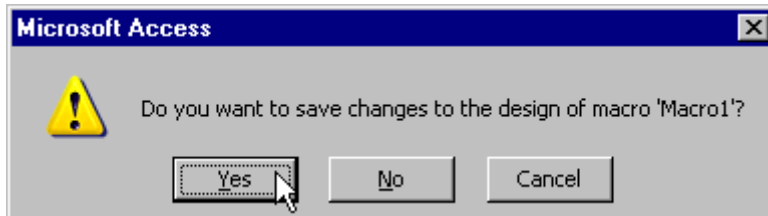
- Once the Object Type is selected, you can select an **Object Name**. Access will list all available Forms in the current database:



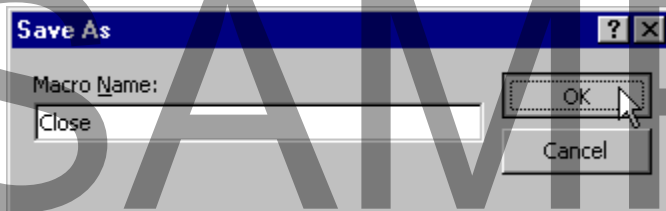
- The last argument is the **Save** option (required). Here, you can select what type of actions Access should perform at the end:



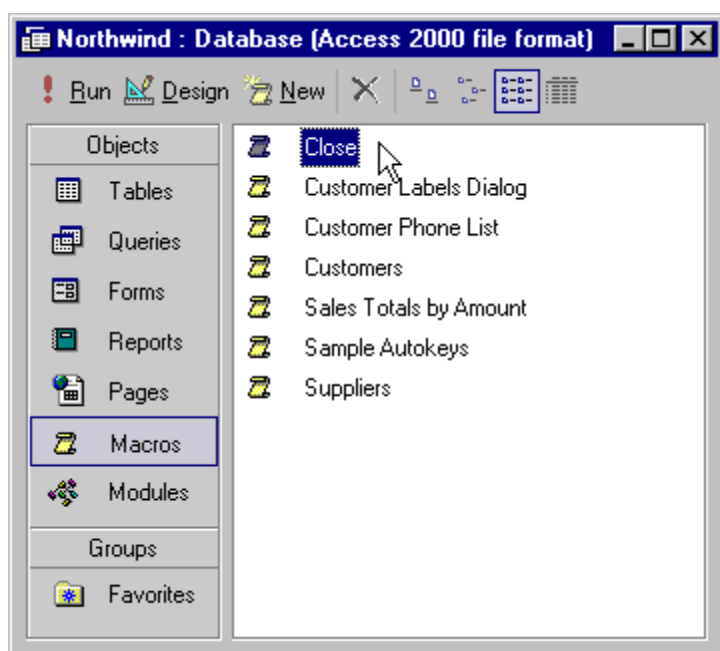
- **Note:** The **information** pane at the bottom-right area of the Macro window displays information about the part of the macro sheet that is currently active. If you need more information, press the **F1** key for help.
- Close the Macro window and click **Yes** to save changes for your new macro:



- In the **Save As** dialog box, enter the name in the **Macro Name** field, then click **OK**:



- The new macro is now available in the **Macros** tab of the **Database** window:



Testing and Debugging Macros

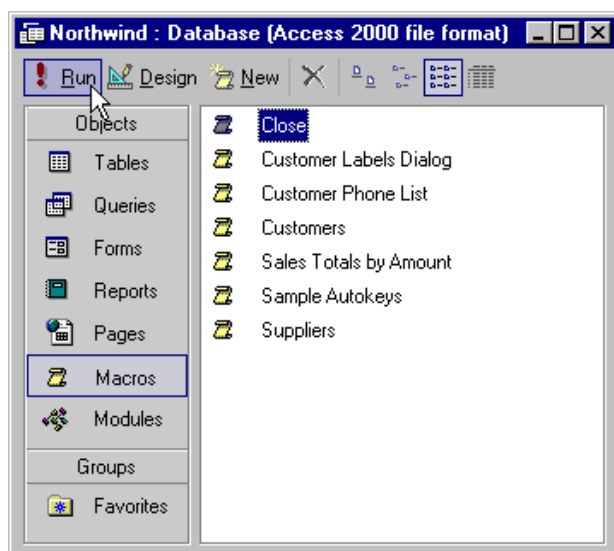
Running a Macro

- Once you have created a macro, you will have to run it and see if it works.
- You can run the whole macro at once, or you can run step-by-step through the macro.
- If you get an error while running a macro, make sure you run through the macro step-by-step to find where the error occurs, and fix it.

To run a macro:

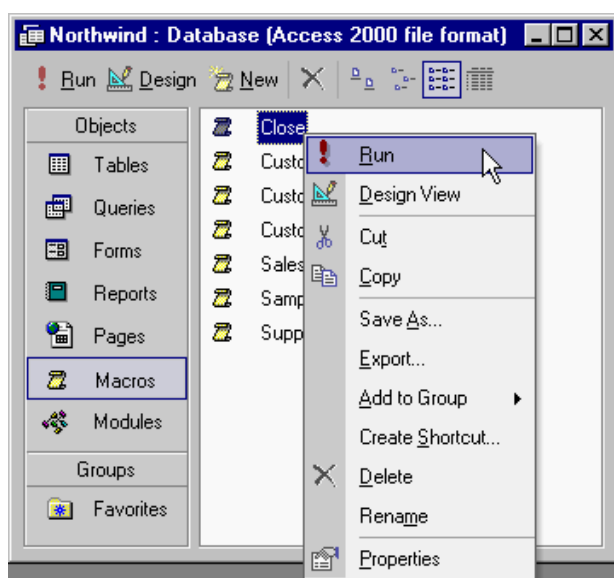
- Select the macro name from the **Macros** page of the **Database** window.
- Click **Run** button:

SAMPLE



OR double-click the macro name

OR right-click the macro name and choose **Run** from the popup menu:

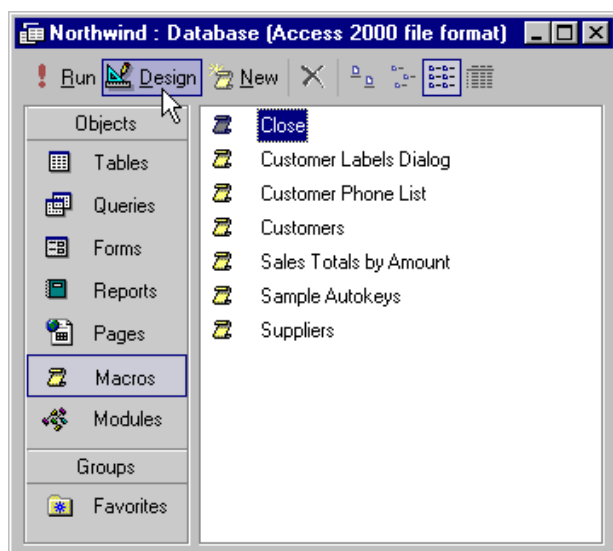


Stepping Through a Macro

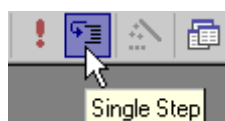
- If you have received an error while running a macro, you will have to step through the macro in order to find the error and fix it.

To use the Single Step method to run a macro:

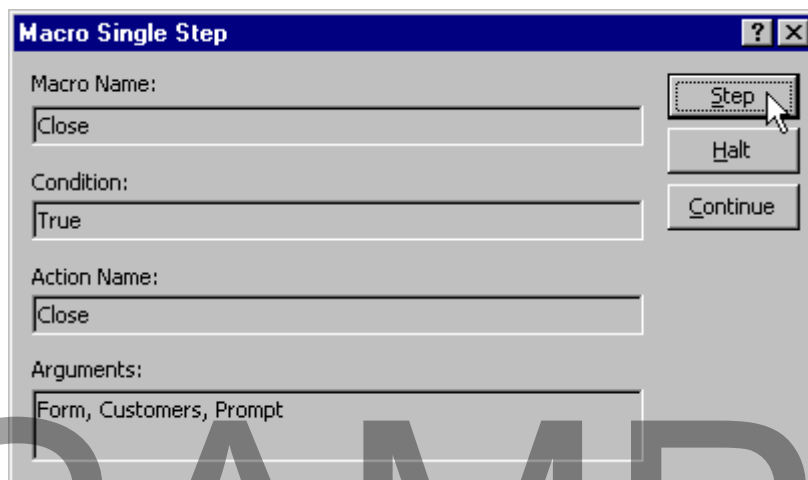
- Select the macro name from the **Macros** page of the **Database** window.
- Click **Design** button to open the **Macro** window:



- While in the **Macro** window, click on the **Single Step** button:



- This will open the **Macro Single Step** dialog box:



- The **Step** button moves to the next action, if there are more actions in the macro. This is the default option.
- The **Halt** button stops macro execution.
- The **Continue** button stops the Single Step mode and runs the rest of the macro without stopping.

Modifying Macros

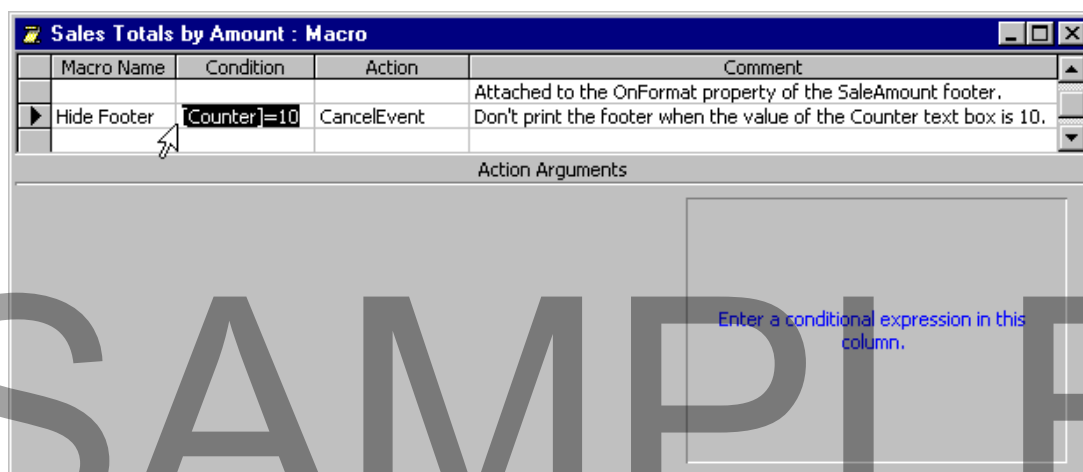
Modifying a Macro

- Select the name of the macro you want to modify from the **Macros** page of the **Database** window.
- Click **Design** button to open the **Macro** window.
- Use the **Insert Rows** toolbar button or the **Insert** menu command to add actions.
- Use the **Delete Rows** toolbar button or the **Edit** menu command to delete actions.
- Use the **Cut**, **Copy**, and **Paste** operations to edit a macro.
- Use the **Undo** button to reverse the recent changes.
- When you have finished editing the macro, **Save** it again.
- **Run** and test your newly saved macro to make sure it works properly.

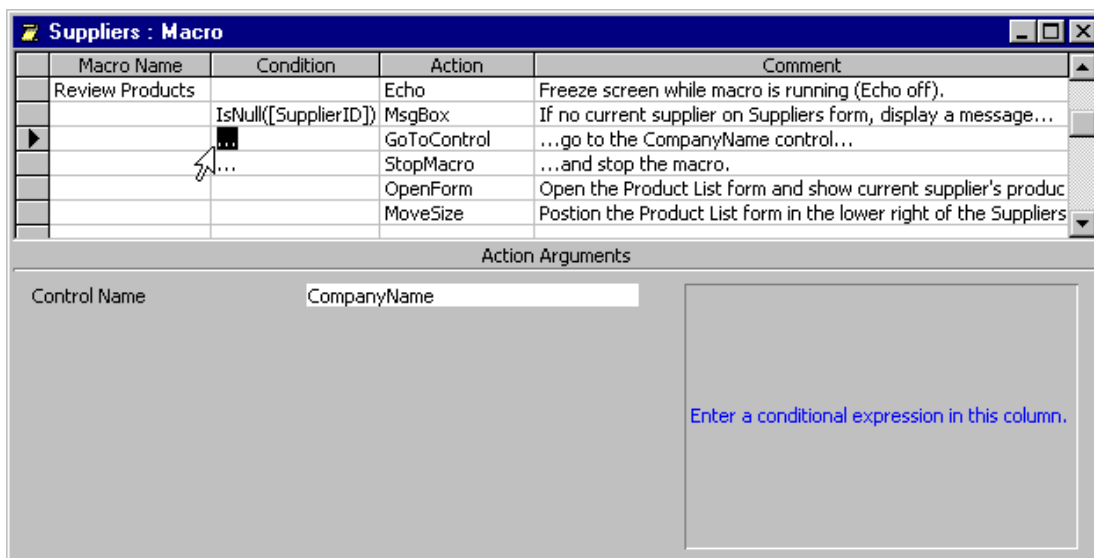
Conditional Programming in Macros

Adding Conditions to Macros

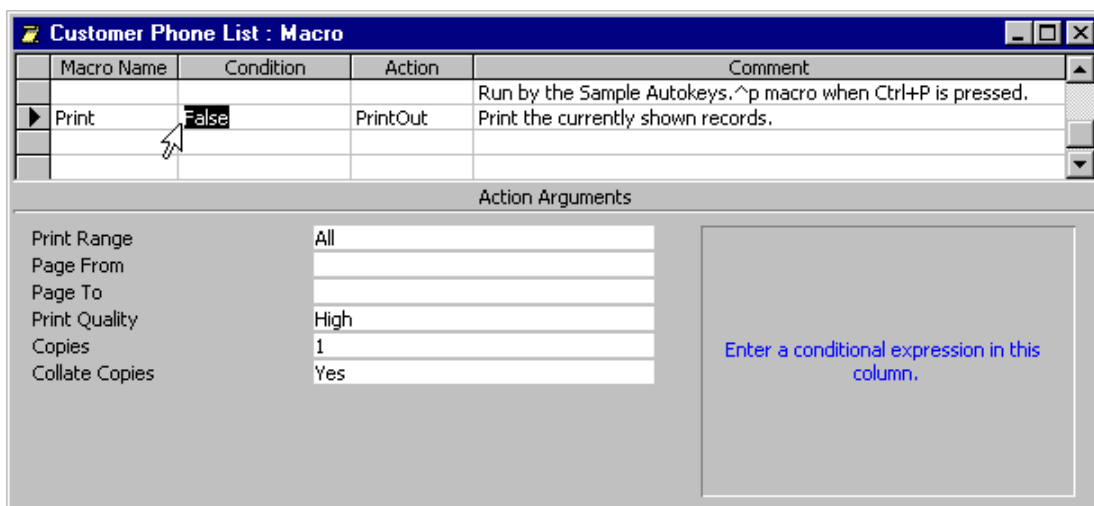
- By adding conditions to macros, you can specify if and when actions will happen.
- Conditions mean that if the condition is true, then the action(s) will be performed. If the condition is not true, the macro will skip to the next action – but only if there is one.
- Macro conditions are added in the **Condition** column on the macro sheet:



- By default, the condition only applies to the action on the same row in the macro sheet.
- If the condition is not met, the next action (i.e. the next row) will be executed.
- To extend the condition to the next action, you must enter the ellipsis (...) in the **Condition** column of the next row:



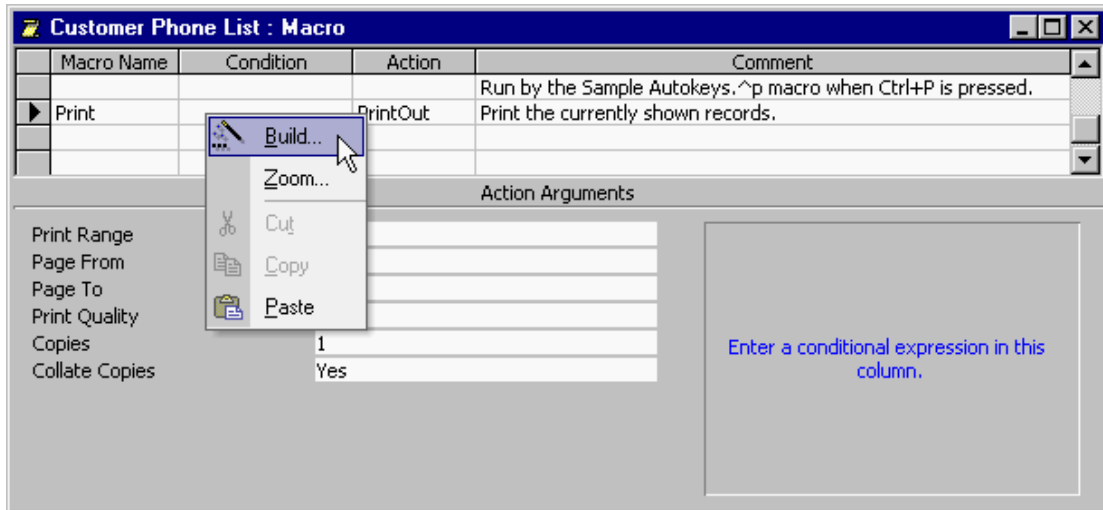
- When debugging a macro, you can temporarily disable an action by entering **False** in the **Condition** column:



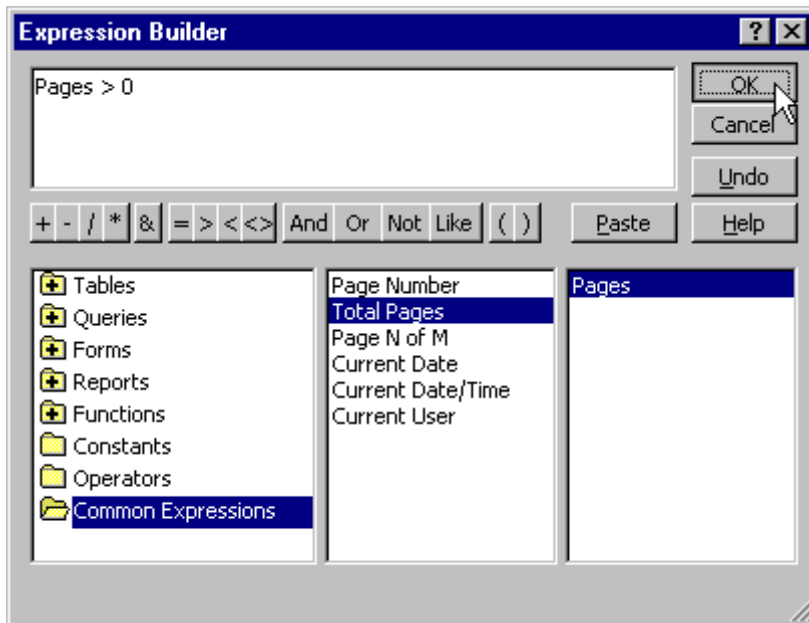
Using the Expression Builder to create Conditions

- Right-click inside the **Condition** field where you want to create a condition, and select **Build** from the popup menu:

SAMPLE

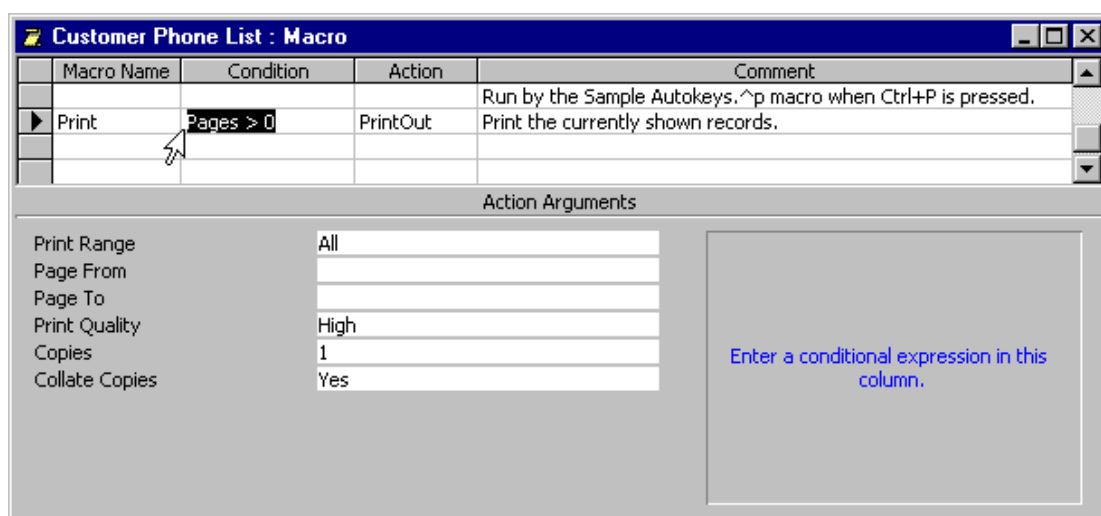


- In the **Expression Builder** dialog box, create the logical expression (for example if the Total Pages number is greater than 0) and click **OK**:



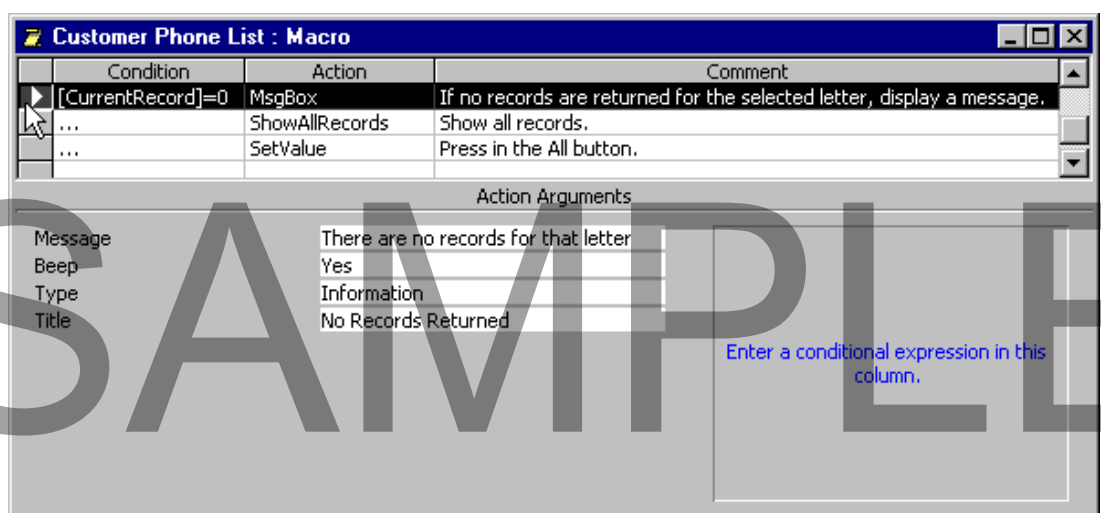
SAMPLE

- The new expression will be displayed in the **Condition** field:



Running Macros with Conditions

- When running macros with conditions, Access evaluates each condition and does each of the following.
- If condition is **TRUE**, Access runs the action on that row first, then runs all following actions that have an ellipsis (...) in the Condition column. Access then runs any additional actions that have blank conditions until it encounters another condition, comes across a macro name in the Macro Name column, or reaches the end of the macro.
- If a condition is **FALSE**, Access ignores the action on that row and any subsequent actions that have an ellipsis (...) in the Condition column. Access then moves to the next condition, if there is one:

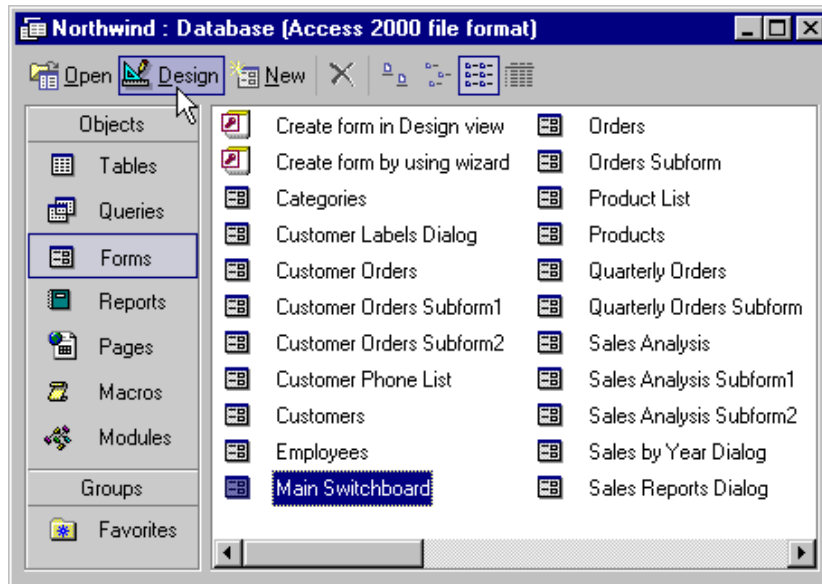


Note: The **MsgBox** action has a condition **[CurrentRecord]=0**, meaning if no records are returned for the selected letter, display a message. Then Access goes to the next action with an ellipsis (...) **ShowAllRecords** that will show all records. Then Access continues to the next action with an ellipsis (...) **SetValue** that will press in the All button.

Adding Macros to Forms

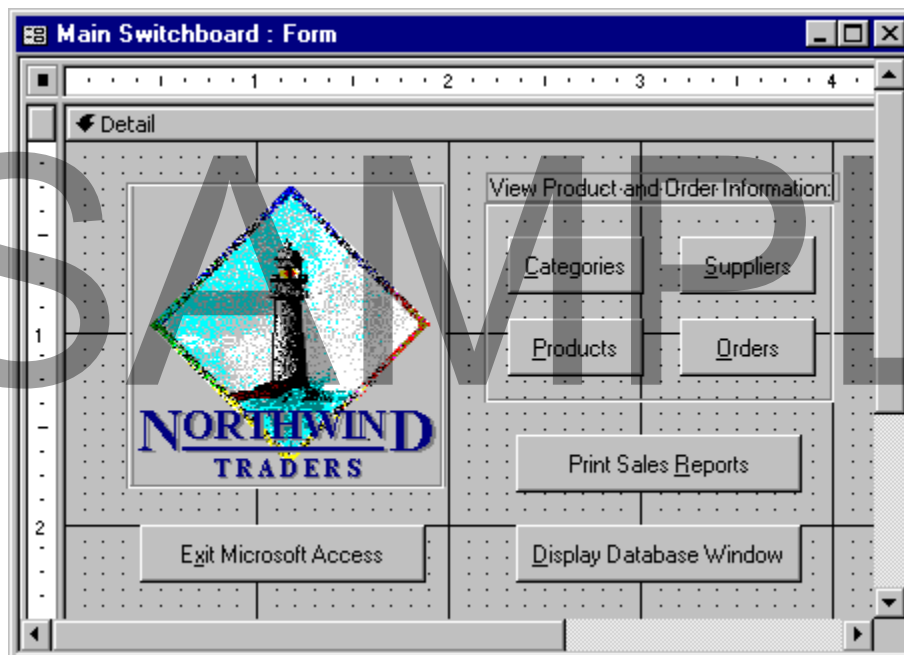
Attaching Macros to a Form

- First select the form in the **Forms** page of the **Database** window and click on the **Design** button:



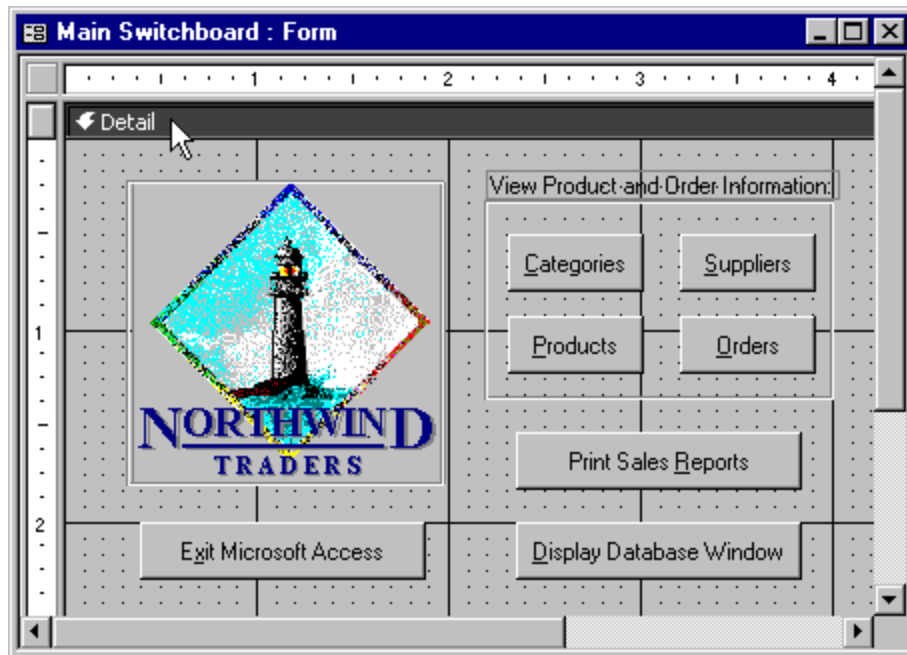
To attach the macro to a form event:

- From the main menu, choose **Edit > Select Form** **OR** press the **Ctrl + R** key combination.
- The **Form** is selected:



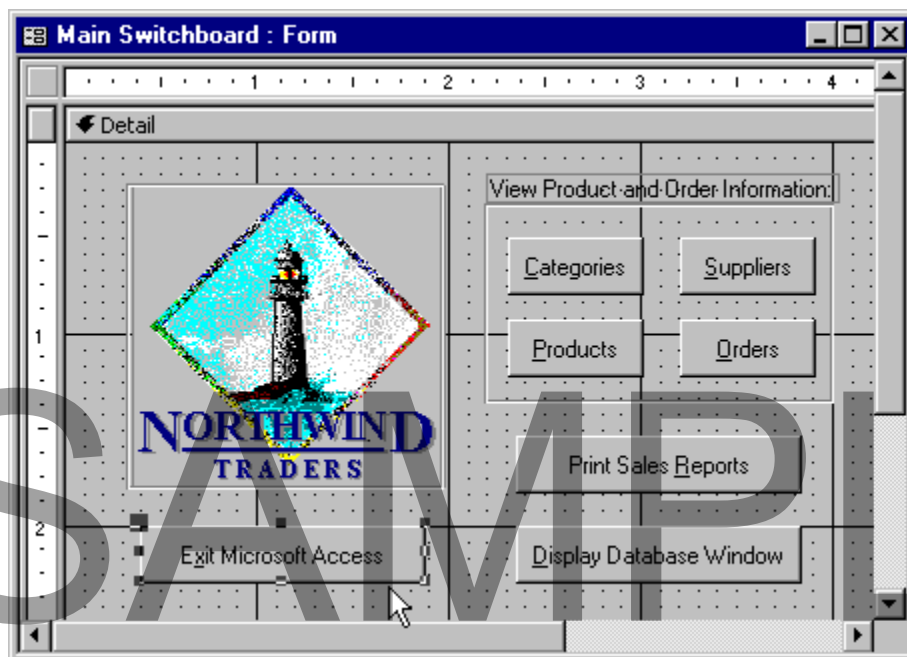
To attach the macro to a section of the form :

- Click the section selector (for example, Detail section):



To attach the macro to a control in the form :

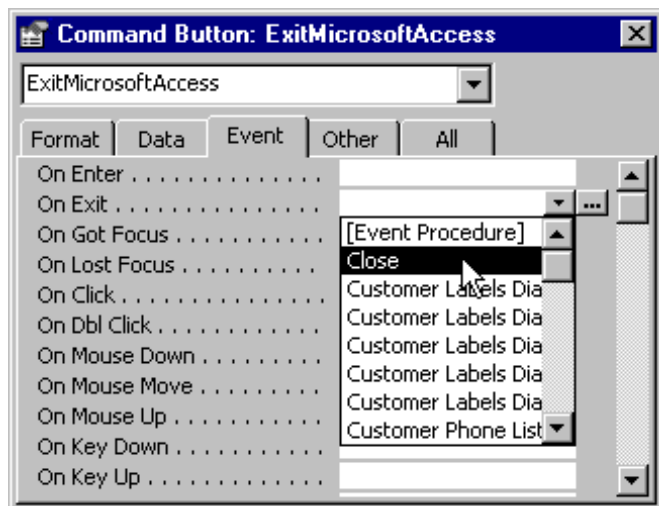
- Select the control (for example, Exit Microsoft Access button):



To attach the macro to the selected form property:

- From the main menu, choose **View > Properties**
OR press the **F4** key
OR right-click on the selection and select **Properties** from the popup menu (available for section or control objects only).

- Click the **Event** tab to view a list of events that can occur for the selected object.
- Select the event you want to run the macro (for example, On Exit).
- Choose the **Macro** name from the drop-down list of available macros (for example, Close):



- **Save** and close the form design.

Adding Macros to Reports

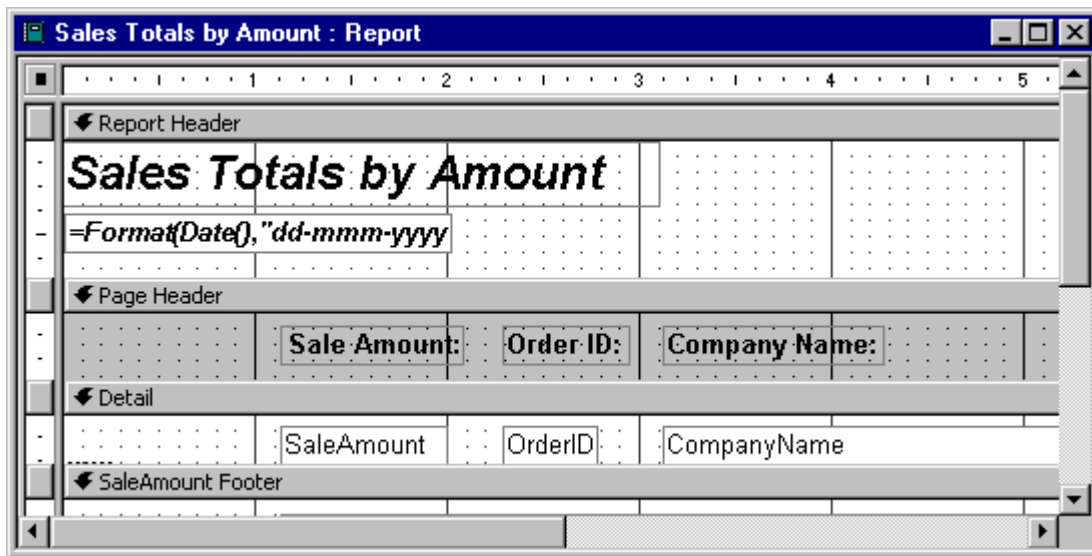
Attaching Macros to a Report

- Select the form in the **Reports** tab of the **Database** window and click on the **Design** button:



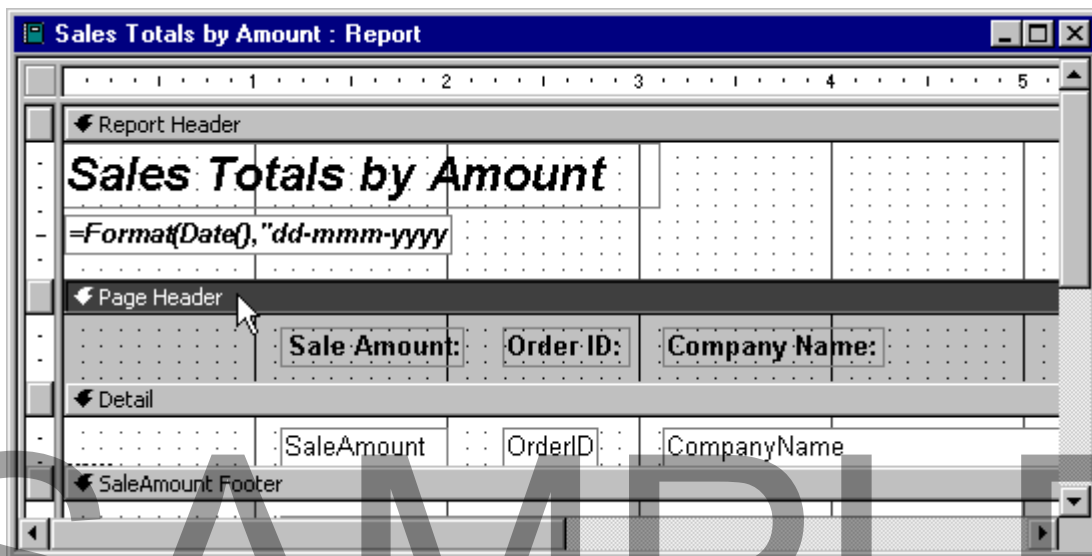
To attach the macro to a report event:

- From the main menu, choose **Edit > Select Report**
OR press the **Ctrl + R** key combination.
- The **Report** is selected:



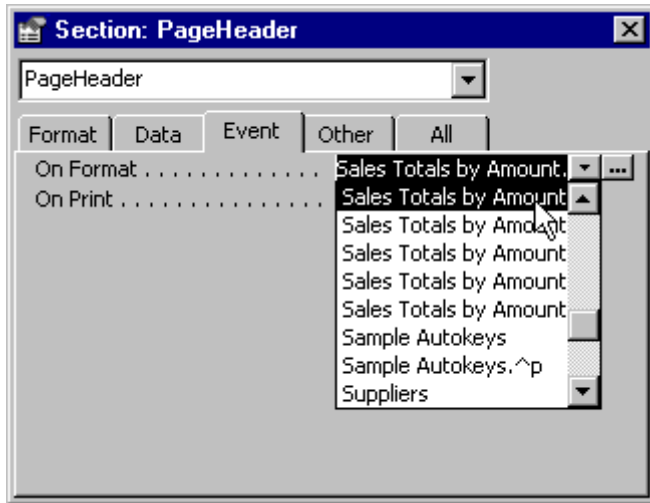
To attach the macro to a section of the report:

- Click the section selector (for example, Page Header section):



To attach the macro to the selected form property:

- From the main menu, choose **View > Properties**
OR press the **F4** key
OR right-click on the selection and select **Properties** from the popup menu (available for the section or control objects only).
- Click the **Event** tab to view a list of events that can occur for the selected object.
- Select the event you want to run the macro (for example, On Format).
- Choose the **Macro** name from the drop-down list of available macros (for example, Sales Totals by Amount):

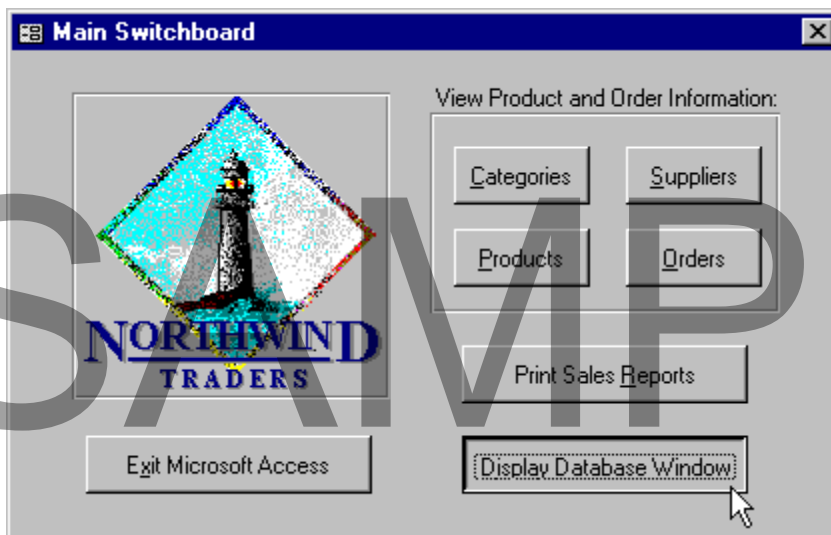


- **Save** and close the report design.

Filtering Data

Filtering Records

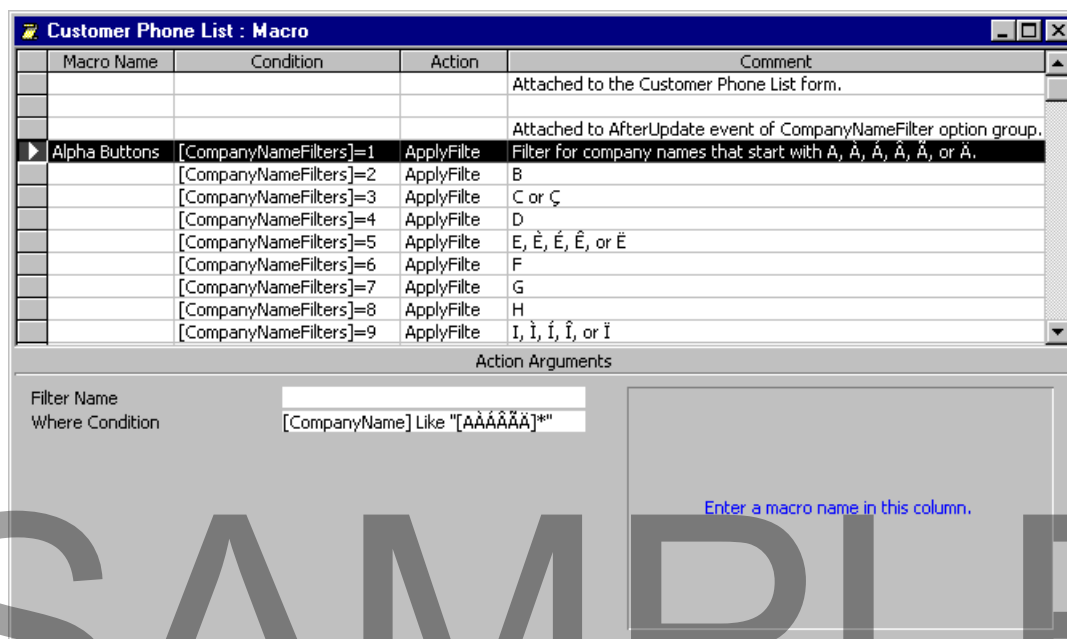
- The Northwind sample database contains a macro that alphabetically filters the records in the Customer Phone List form.
- Open the Northwind database by selecting **Help > Sample Databases > Northwind Sample Database** from the main menu.
- In the **Main Switchboard** window, click on the **Display Database Window** button:



- In the **Database** window, click on the **Macros** tab, then select the **Customer Phone List** macro and click on the **Design** button:



- This will open the macro sheet:



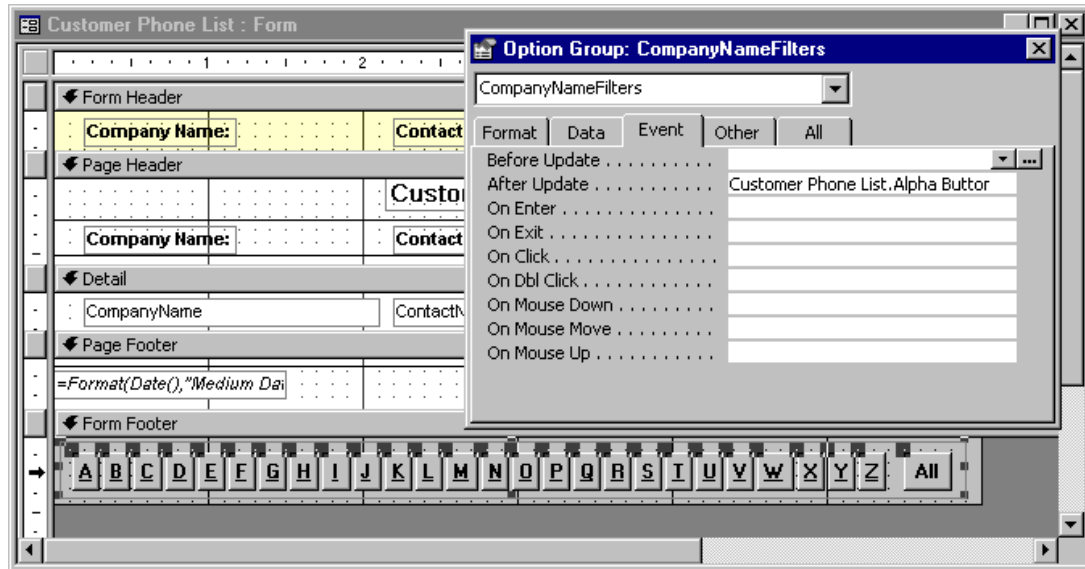
- In the Comment column, you can see that this macro is attached to the Customer Phone List form. In order to see how the macro works, we should run it first.
- In order to run the macro, we have to open the form attached to the macro.
- Go back to the **Database** window, click on the **Forms** tab, and select **Customer Phone List**:



- Double-click on the form name to open it:

Company Name:	Contact:	Phone:	Fax:
Alfreds Futterkiste	Maria Anders	030-0074321	030-0076545
Ana Trujillo Emparedados y helados	Ana Trujillo	(5) 555-4729	(5) 555-3745
Antonio Moreno Taquería	Antonio Moreno	(5) 555-3932	
Around the Horn	Thomas Hardy	(171) 555-7788	(171) 555-8750
Berglunds snabbköp	Christina Berglund	0921-12 34 65	0921-12 34 67
Blauer See Delikatessen	Hanna Moos	0621-08460	0621-08924
Blondel père et fils	Frédérique Citeaux	88.60.15.31	88.60.15.32
Bólido Comidas preparadas	Martín Sommer	(91) 555 22 82	(91) 555 91 99
Bon app'	Laurence Lebihan	91.24.45.40	91.24.45.41
Bottom-Dollar Markets	Elizabeth Lincoln	(604) 555-4729	(604) 555-3745

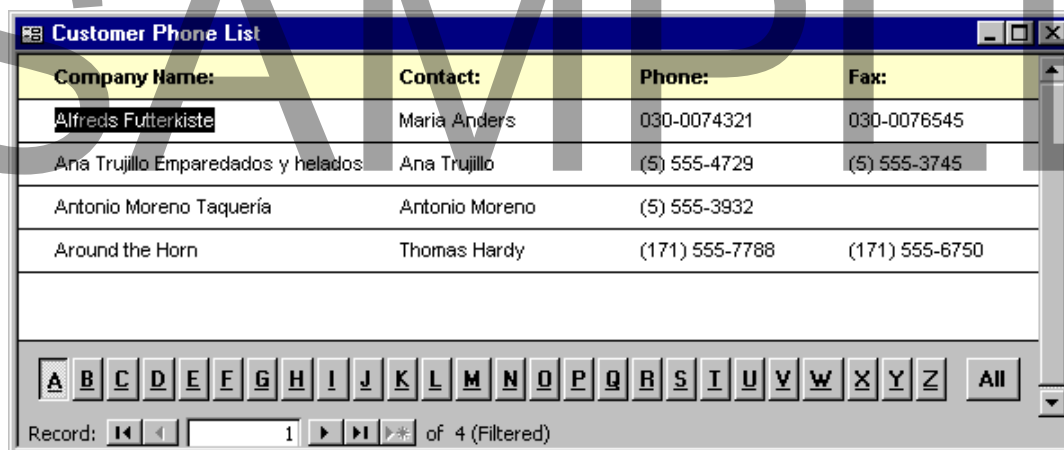
- As you can see, the form has Alphabet buttons that will filter data when pressed. Only the company names that start with a specified letter of the alphabet will be displayed.
- Click on the **Design** button to see the underlined programming for this form:



- The row of Alphabetic buttons are actually an **Option Group** control named **CompanyNameFilters**, which returns a value depending which button is selected.
- If you open the **Properties** for this control, you will see that the macro **Customer Phone List.Alpha Button** is attached to the **After Update** event. This means that the macro will run when the user clicks one of the buttons in the option group.
- Go back to the **Customer Phone List** macro sheet to see how this is done:

Macro Name	Condition	Action	Comment
			Attached to the Customer Phone List form.
			Attached to AfterUpdate event of CompanyNameFilter option group.
Alpha Buttons	[CompanyNameFilters]=1	ApplyFilter	Filter for company names that start with A, Å, Ä, Å, Ä, or Å.

- The **Alpha Buttons** macro applies a filter to the list, based on which button is pressed.
- For example, if the **A** button is pressed, the macro will filter for company names that start with Å, Ä, Å, Ä, Å, or Ä letters.
- Pressing on the **A** button will result in this company list:



- Now go back to the **Customer Phone List** macro sheet and scroll down to the last rows. In the **GoToControl** action row, the stated condition is that if the current record count is greater than 0, the records are displayed and the macro stops. The ellipsis (...) in the **StopMacro** action row carries over the condition from the row above it:

Macro Name	Condition	Action	Comment
	[CompanyNameFilters]=24	ApplyFilter	X
	[CompanyNameFilters]=25	ApplyFilter	Y, Ý, or ÿ
	[CompanyNameFilters]=26	ApplyFilter	Z, Æ, Ø, or Å
	[CompanyNameFilters]=27	ShowAllRecord	Show all records.
	[CurrentRecord]>0	GoToControl	If records are returned for the selected letter, go to the CompanyName control
	...	StopMacro	Stop the macro.
	[CurrentRecord]=0	MsgBox	If no records are returned for the selected letter, display a message.
	...	ShowAllRecord	Show all records.
	...	SetValue	Press in the All button.

- However, if no records are returned by the filter (based on the condition if the current record count equal to 0), the message will be displayed in the **MsgBox** action. When the user clicks **OK** to close the message window, all records are displayed on the screen, as the ellipsis (...) in the **ShowAllRecord** action row carries over the condition from the row above it. Furthermore, the next action row containing ellipsis (...) **SetValue**, will press in the **All** button in the form:

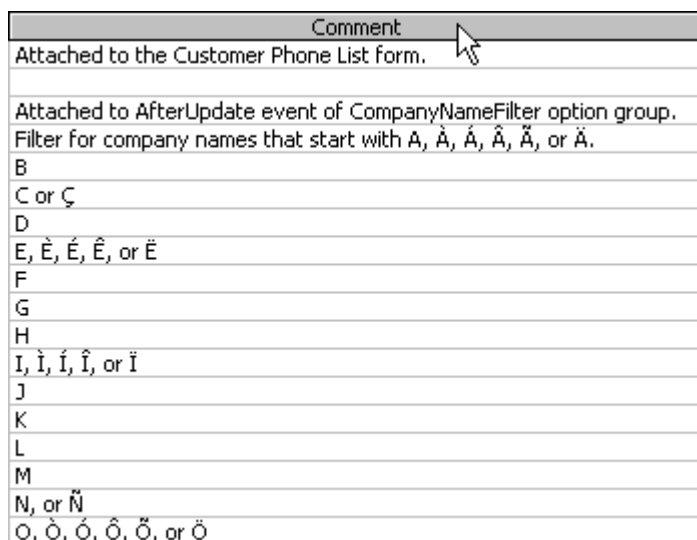
Macro Name	Condition	Action	Comment
	[CompanyNameFilters]=24	ApplyFilter	X
	[CompanyNameFilters]=25	ApplyFilter	Y, Ý, or ÿ
	[CompanyNameFilters]=26	ApplyFilter	Z, Æ, Ø, or Å
	[CompanyNameFilters]=27	ShowAllRecord	Show all records.
	[CurrentRecord]>0	GoToControl	If records are returned for the selected letter, go to the CompanyName control
	...	StopMacro	Stop the macro.
	[CurrentRecord]=0	MsgBox	If no records are returned for the selected letter, display a message.
	...	ShowAllRecord	Show all records.
	...	SetValue	Press in the All button.

Documenting Macros

Commenting Macros

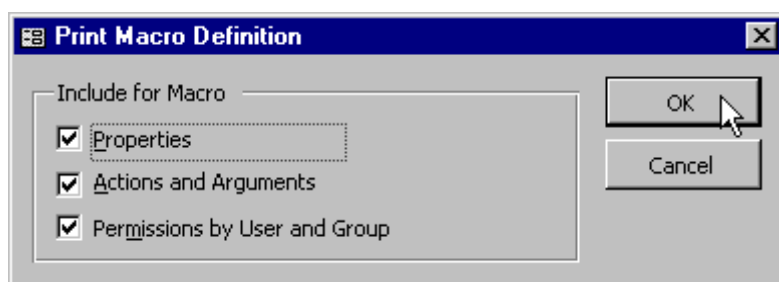
- To make programming and future updates easier, make sure you comment macros using the **Comment** column in the macro sheet:

SAMPLE



Printing Macro Definitions

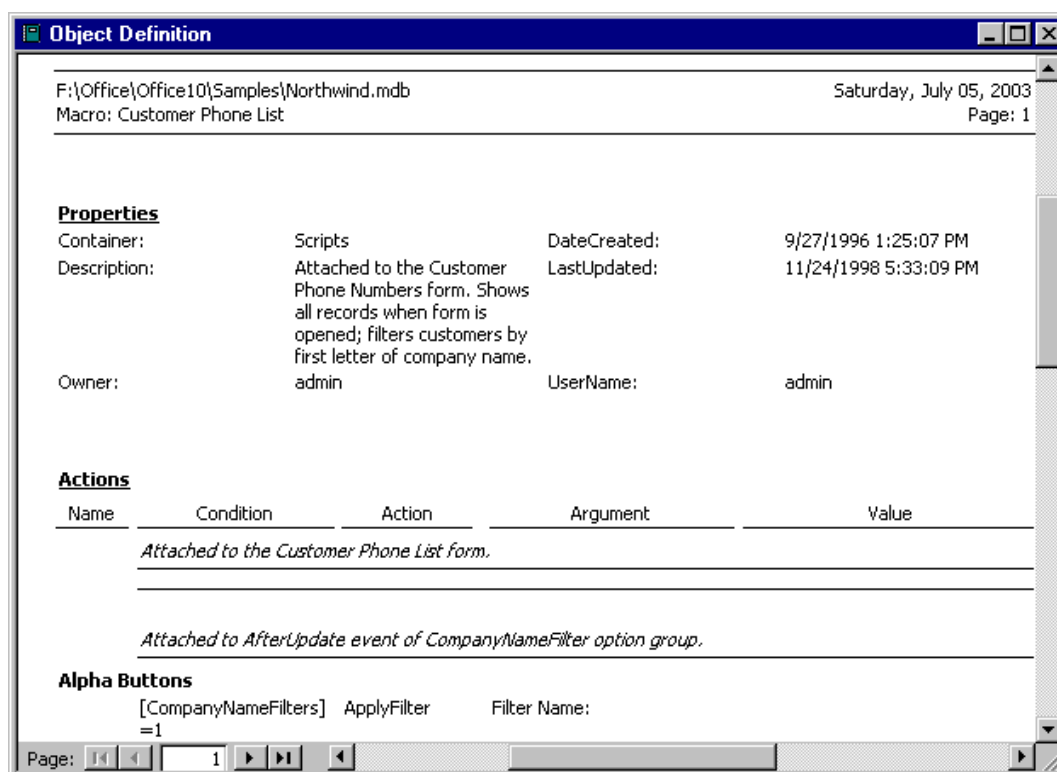
- In the **Macros** page of the **Database** window, select the macro name.
- From the main menu, choose **File > Print** *OR* press the **Ctrl + P** key combination.
- In the **Print Macro Definition** dialog box, select what info you want to print and click **OK**:



- **Properties** include the container, date created, date of last update, owner, and user.
- **Actions and Arguments** include all the actions with their conditions, as well as values for all arguments.
- **Permissions by User and Group** include user permissions and group permissions.

Viewing Macro Definitions

- In the **Macros** page of the **Database** window, select the macro name.
- From the main menu, choose **File > Print Preview** and make the same selections as for the Print:



Review Questions

How would you:

- Understand Macros?
- Use the Macro Window Toolbar?
- Create New Macros?
- Run a Macro?
- Step Through a Macro?
- Modify a Macro?
- Add Conditions to Macros?
- Use the Expression Builder to Create Conditions?
- Run Macros with Conditions?
- Attach Macros to a Form?
- Attach Macros to a Report?
- Filter Records?
- Comment Macros?
- Print Macro Definitions?
- View Macro Definitions?

Programming Access Using Visual Basic

When you have completed this learning module you will have seen how to:

- Use Visual Basic Modules
- Convert Macros to Visual Basic Code
- Understand Modules
- Create Modules
- Understand Module Declarations
- Understand Procedures
- Use Naming Rules
- Declare Variables
- Set Variable Scope
- Declare Constants
- Use Methods
- Use Arguments
- Customize the Visual Basic Editor Window
- Set the Visual Basic Editor Options
- Use Microsoft Visual Basic Help
- Get Visual Basic Syntax Help

Using Macros versus Visual Basic

- In this manual, we will cover only the basics of Visual Basic programming regarding its use in Access 2002.
- If you want to learn more about Visual Basic, refer to one of the numerous manuals and books on this subject.

Using Visual Basic Modules

- In the previous chapter, we have covered the use of Macros. Another way to program Access is writing **Visual Basic** modules.
- Certain procedures cannot be created in Macros, you must use Visual Basic to write them.
- Examples of Visual Basic usages:
 - Error handling
 - Repetitive looping
 - Custom functions
 - Optimized performance, etc

Converting Macros to Visual Basic Code

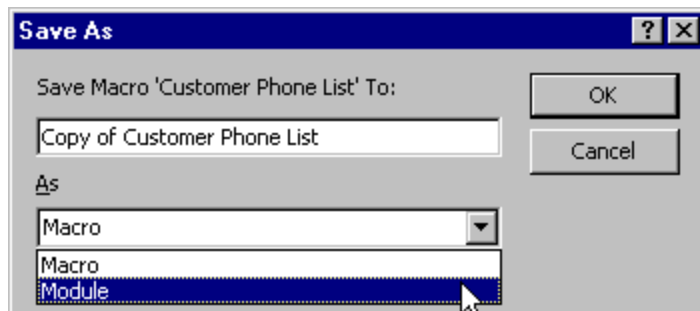
- There are two methods for converting Macros to Visual Basic (VB) code:
- Converting from the Macros page of the **Database window** – this is used when you want to make VB code available to the whole database.

- Converting from the Form or Report **Design view** – this is used when you want to store VB code with a form or report.

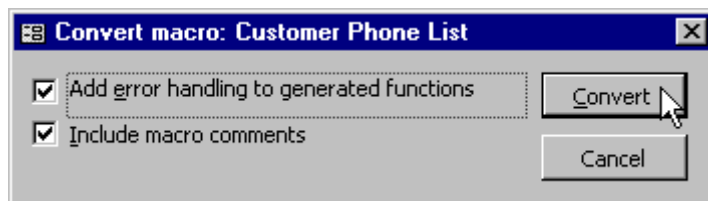
To convert a Macro from the Database Window :

- Click on the **Macros** tab in the Database window.
- Select a macro you want to convert to VB.
- From the main menu, choose **Tools > Macro > Convert Macros to Visual Basic**

OR from the main menu, choose **File > Save As** and in the **Save As** dialog box, select **Module** under the **As** filed. Click **OK** to continue:

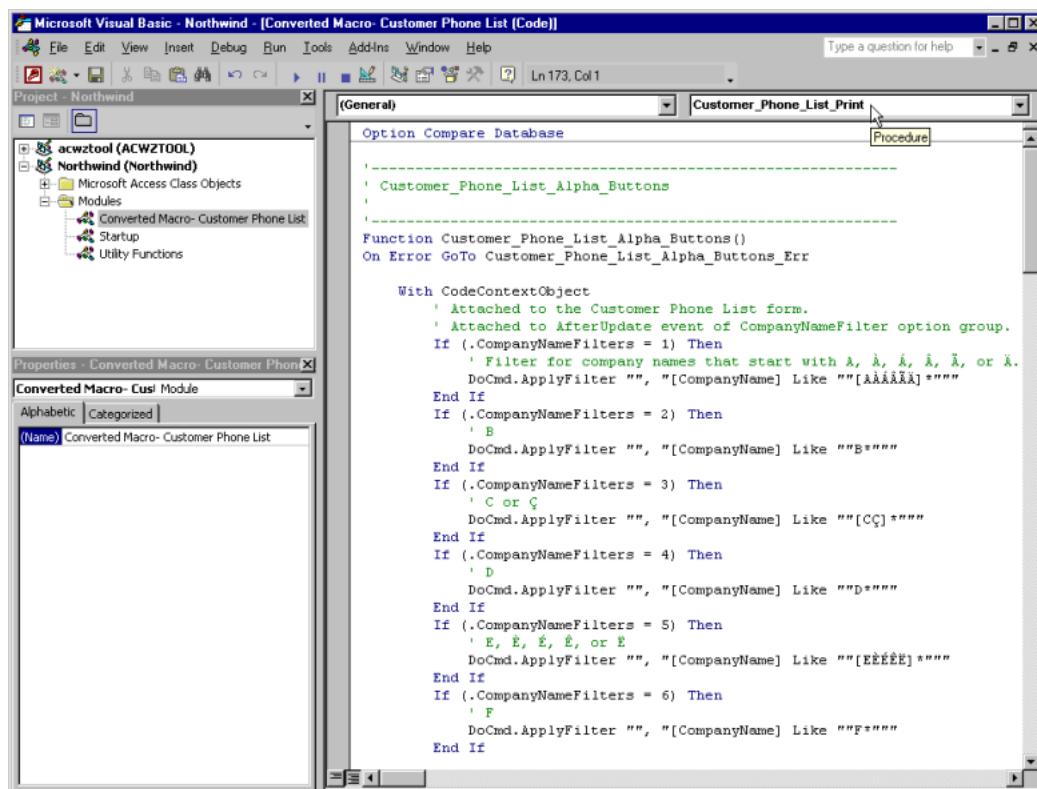


- This will open up the **Convert macro: *macro_name*** dialog box:



- Check the Add error handling to generated functions and Include macro comments options and click Convert.
- The **Convert macros to Visual Basic** message box shows the conversion is finished.
- This will create **Converted Macro – *macro_name*** module, listed under the **Modules** folder in the Visual Basic Editor window. Click on the module to see the code:

SAMPLE



- Scroll to the end of the code window to view the whole Visual Basic code:

```

'-----
' Customer_Phone_List_Print
'-----

Function Customer_Phone_List_Print()
On Error GoTo Customer_Phone_List_Print_Err

' Run by the Sample Autokeys.^p macro when Ctrl+P is pressed.
If (False) Then
' Print the currently shown records.
DoCmd.PrintOut acPrintAll, , , acHigh, 1, True
End If

Customer_Phone_List_Print_Exit:
Exit Function

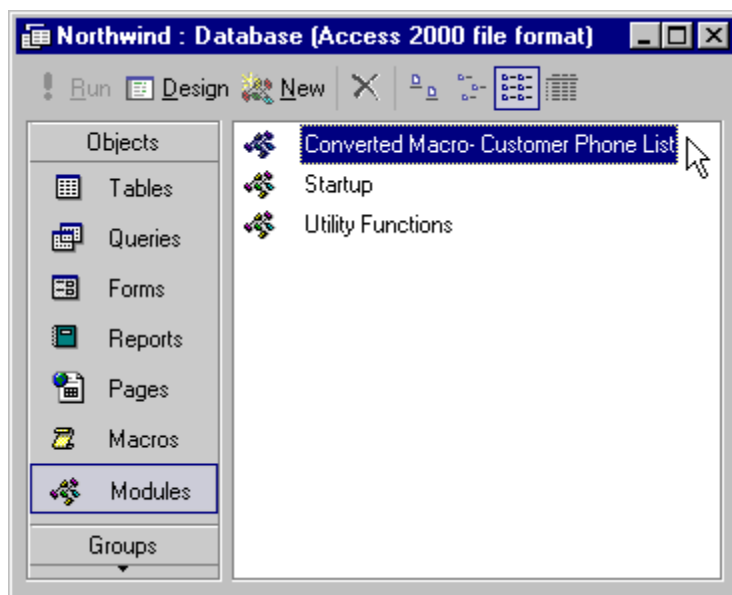
Customer_Phone_List_Print_Err:
MsgBox Error$
Resume Customer_Phone_List_Print_Exit

End Function

```

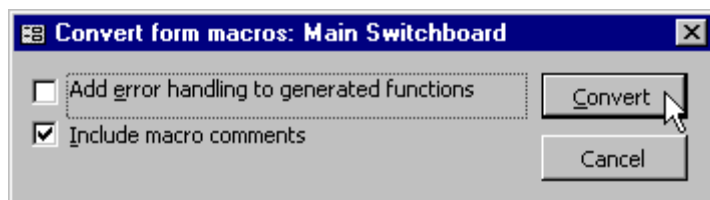
- The **Function** statement represents the global module that is available to the entire database.
- The **DoCmd** statement performs macro action in a Visual Basic procedure.
- **Apostrophe (')** marks the comment line.

- Newly created module is also listed in the **Modules** tab of the Database window:



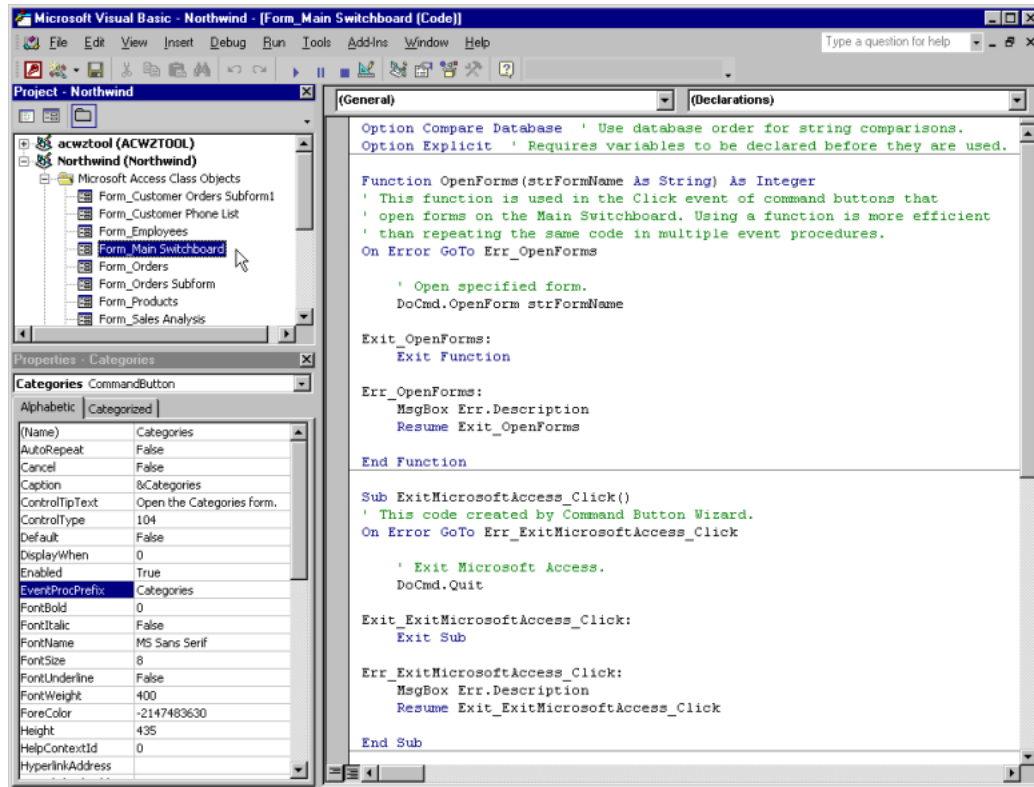
To convert Form's Macro from the Design View :

- Click on the **Forms** tab in the Database window.
- Select a form whose macro you want to convert to VB.
- Click on the **Design** view button.
- From the main menu, choose **Tools > Macro > Convert Form's Macros to Visual Basic**.
- This will open up the **Convert form macros: *macro_name*** dialog box:



- Uncheck **Add error handling to generated functions** and check **Include macro comments** options and click Convert.
- The **Convert macros to Visual Basic** message box shows the conversion is finished.
- Click on the **Code** toolbar button to open the **Visual Basic Editor** window with the converted macros code:





- Scroll to the end of the code window to view the whole Visual Basic code:

```
Sub ExitMicrosoftAccess_Click()
' This code created by Command Button Wizard.
On Error GoTo Err_ExitMicrosoftAccess_Click

' Exit Microsoft Access.
DoCmd.Quit

Exit_ExitMicrosoftAccess_Click:
Exit Sub

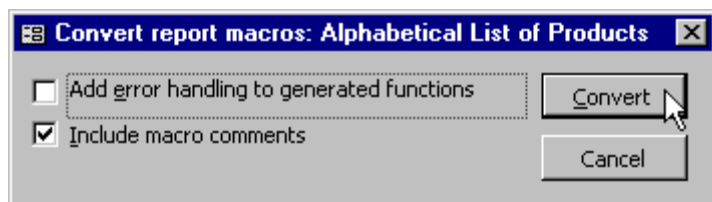
Err_ExitMicrosoftAccess_Click:
MsgBox Err.Description
Resume Exit_ExitMicrosoftAccess_Click

End Sub
```

- The **Sub** statement represents the local code, stored with a form.
- The **DoCmd** statement performs macro action in a Visual Basic procedure.
- The **Apostrophe (')** marks the comment line.

To convert Report's Macro from the Design View:

- Click on the **Reports** tab in the Database window.
- Select a report whose macro you want to convert to VB.
- Click on the **Design** view button.
- From the main menu, choose **Tools > Macro > Convert Report's Macros to Visual Basic**.
- This will open up the **Convert report macros: macro_name** dialog box:



- Uncheck **Add error handling to generated functions** and check the **Include macro comments** option and click **Convert**.
- The **Convert macros to Visual Basic** message box shows the conversion is finished.

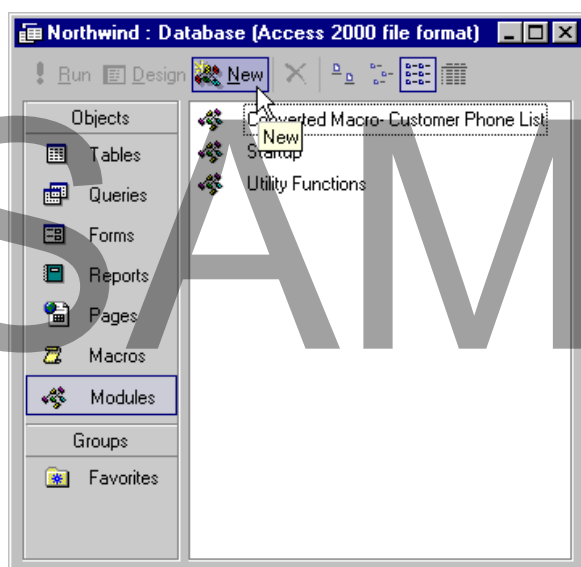
Understanding Visual Basic Concepts

Understanding Modules

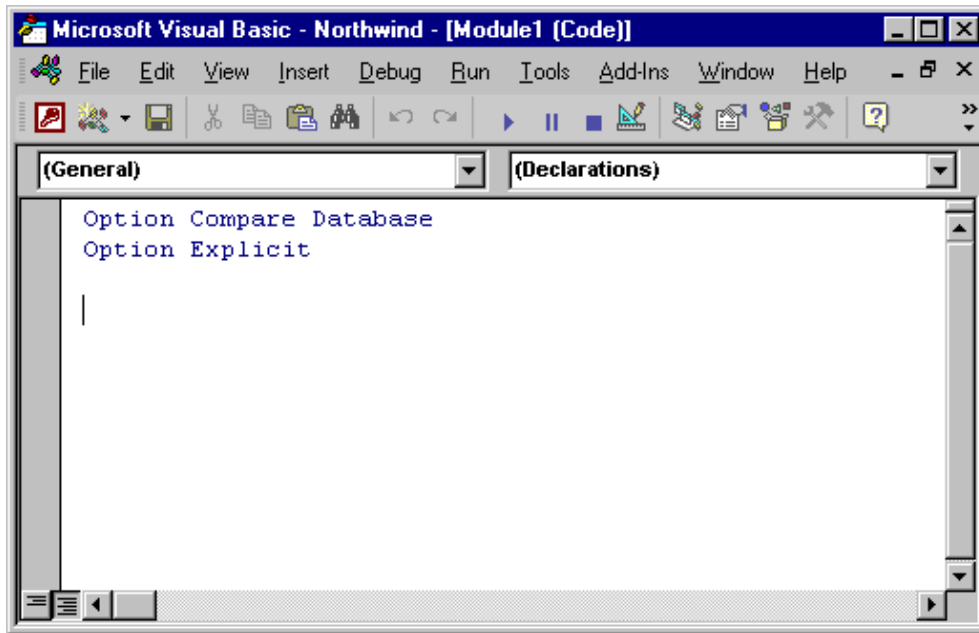
- The Visual Basic module consists of declarations and procedures.
- Modules are used to create event procedures that execute when an event occurs.
- There are two types of modules:
 1. **Module objects** listed in the Modules page of the Database window.
 2. **Form and Report modules** stored together with a form or report containing procedures and functions associated with a single form or report.

Creating Modules

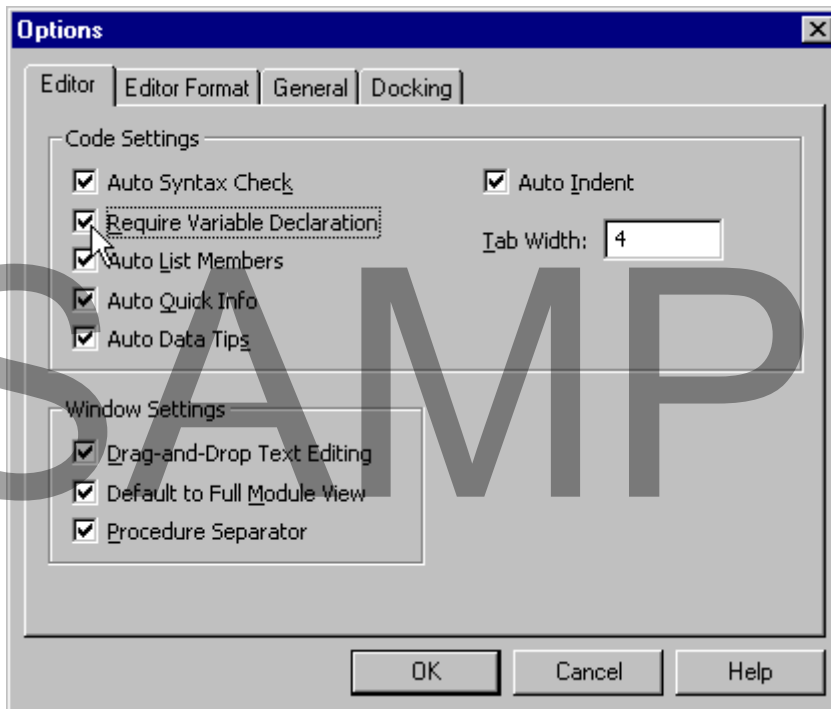
- Click on the **Modules** tab in the Database window.
- Click on the **New** button:



- The new module, named **Module1**, opens up in the **Visual Basic Editor** window:



- A newly created module has two or more sections:
Declarations section
Procedure section(s)
- If you do not see the **Option Explicit** declaration, choose **Tools > Options** from the Visual Basic Editor window main menu.
- In the **Options** dialog box, click on the **Editor** tab and check the **Require Variable Declaration** option. Click **OK** to continue:



Understanding Module Declarations

- Declarations are listed in the first section of a module.
- They set requirements and defaults on the module-level.
- There are two default module-level declarations:
- The **Option Compare** declaration specifies the default comparison method for string data.
This declaration requires an argument:
 - **Binary** sorts string data based on the internal binary representation of characters.
 - **Text** sorts in a case-insensitive text sort order.
 - **Database** sorts in the same order as the database.
- The **Option Explicit** declaration forces the declaration of variables used in the module.

Understanding Procedures

- Procedures consist of statements and methods.
- A statement is a complete instruction to Visual basic and consists of keywords, operations, variables, constants, and expressions.
- There are three types of statements:
- **Declaration** statements used to name variables, constants or procedures.
- **Assignment** statements used to give a value or an expression to a variable or a constant.
- **Executable** statements used to trigger actions or jump to another location in the procedure.

There are two types of procedures:

- A **sub** procedure is program code that does not return a value. It usually runs as a separate program called by an event in a form or report.
- Example of a sub procedure:

```
Sub sub_name (parameters)
```

```
DoCmd.command_name
```

```
End Sub
```

- A **function** procedure is program code that returns a value as a result of a calculation or comparison.
- Example of a function procedure:

```
Function function_name (parameters) As type
```

```
    calculation
```

```
End Function
```

Using Naming Rules

- In any programming, writing clear code is one of the most important practices. This helps you and others understand and maintain your code.
- In Visual Basic, there are several naming rules and practices:
- The first character must be a letter.
- The name has a maximum of 255 characters.
- Avoid the use of special characters and spaces, periods, exclamation marks, etc.
- Avoid the use of Visual Basic keywords.
- Avoid the use of same names for multiple variables in same procedures.
- Add prefixes to names that can indicate what type of item or variable they are:
 - **con** used to indicate a constant
 - **int** used to indicate an integer variable
 - **dte** used to indicate a date variable
 - **str** used to indicate a string variable
 - **rst** used to indicate a recordset
 - **frm** used to indicate a form

Declaring Variables

- **Variables** are items with unique names that contain data that can be changed during procedure execution.
- You can specify the data type for each variable, or let Access assign the default **Variant** type.
- A **Variant** type variable can contain string, date, time, Boolean, or numeric values.
- The **Dim** statement is used to declare variables:

Dim strName As String

Dim intAge As Integer, dteBirthday As Date, undeclared

- You can declare several variables in the same statement, separated by commas.
- In the above example, we have used prefixes to descriptively name the variables in order to include their data type.
- The last variable has no data type declaration, so the type is Variant.

Setting Variable Scope

- **Scope** of a variable indicates the variable availability to procedures.
- There are three levels of variable scope:

Procedure level is set when the variable is declared with a **Dim** statement in the procedure.

Private module level is set when the variable is declared with a **Private**

statement in the declarations module section. The variable is available only to procedures in that module.

Public module level is set when the variable is declared with a **Public** statement in the declarations module section. The variable is available to all procedures in the application.

Declaring Constants

- **Constants** are items with unique names within which data cannot be changed during procedure execution.
- A constant data type can be a string or number value, another constant, or an expression.
- The **Const** statement is used to declare constants and give them a value:

```
Const conYear As Integer = 1990
Const conTitle As String = "President"
```

- You can declare several constants in the same statement separated by commas.
- In the above example, we have used prefix con to descriptively name the constants instead of their data types.
- You must assign a constant a value at the time of declaration.

Using Methods

- **Methods** are actions that database objects can perform, or procedures that apply to database objects.
- Access 2002 modules can use nearly 160 methods that apply to different database objects.
- Some examples of methods are:
 - Opening or closing a form.
 - Sounding a beep when specific event occurs.
 - Going to a specific field in a form.
 - Filtering records for a report.
 - Printing multiple copies of a report, etc.

- Methods have specific syntax that indicates the object and the action:

```
SetFocus Forms![Customer Phone List]![Company Name].SetFocus
```

- In the example above, the method moves the cursor to the **Company Name** field in the **Customer Phone List** form.

Using Arguments

- **Arguments** are used in the sub or function procedure calls.
- They are optional, depending on the procedure.

- There are two ways you can list arguments: in a procedure order, or by name.
- For example, this is the sub procedure with argument list:

```
Sub GetArguments (strName As String, dteBirthday As Date, intAge As Integer)
```

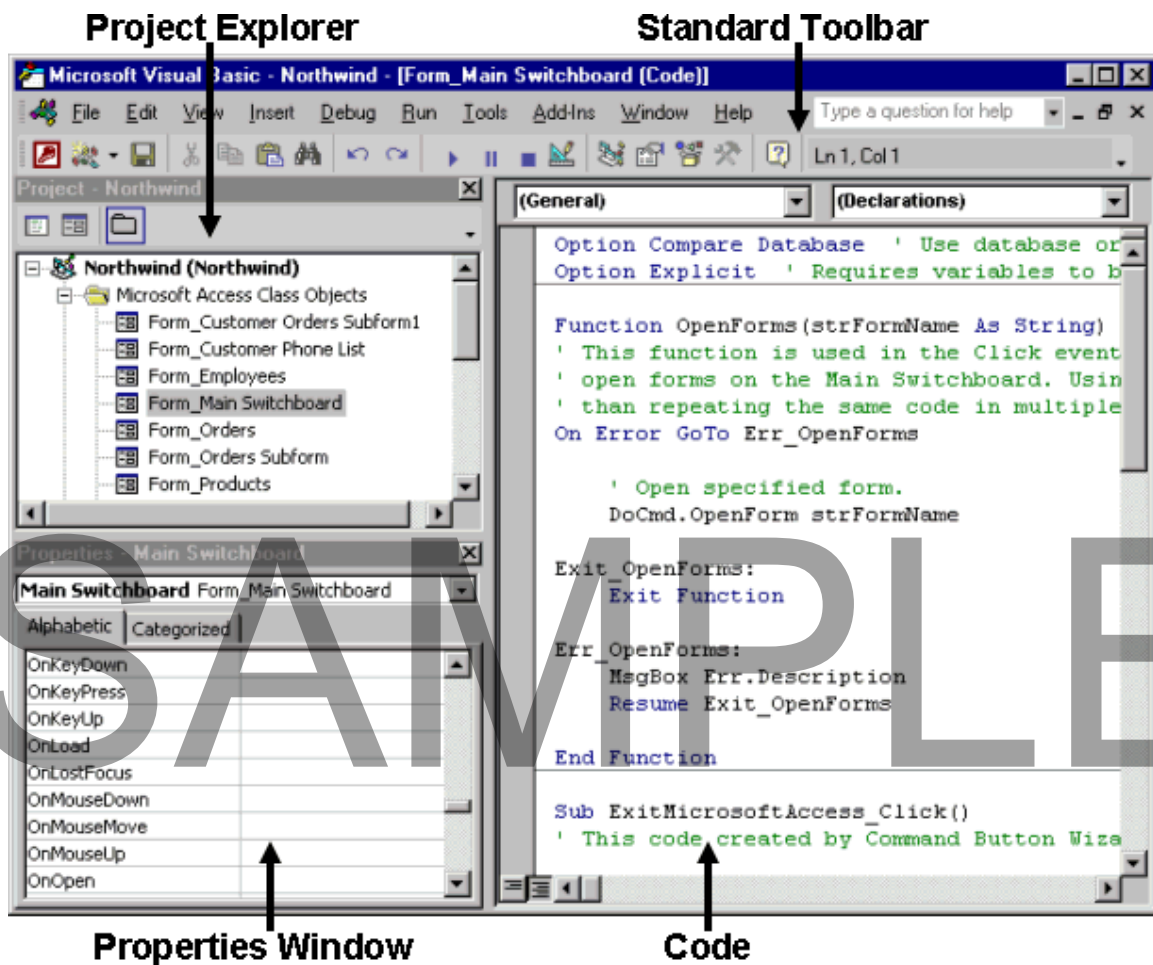
- When you run the procedure, you can list the arguments in the same order as the procedure:

```
GetArguments "John", #11/12/1967#, 36
```

- Or, you can list the arguments using their names – in any order:

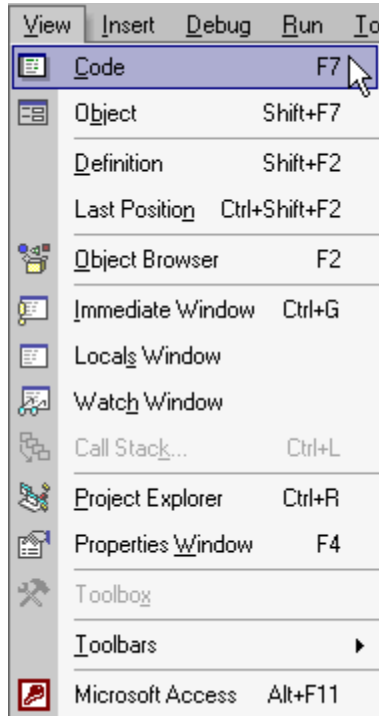
```
GetArguments dteBirthday: =#11/12/1967#, intAge: =36, strName: ="John"
```

Using the Visual Basic Editor Window

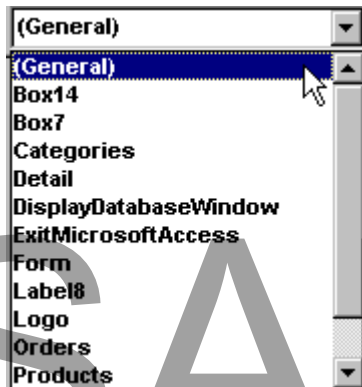


Customizing the Visual Basic Editor Window

- Once you open the **Visual Basic Editor** window, you can change its look.
- From the main menu, choose **View** > then select the elements you want to see in the editor window:



- In the **Code** view, the **Object** list box displays a list of all controls in the form or report:

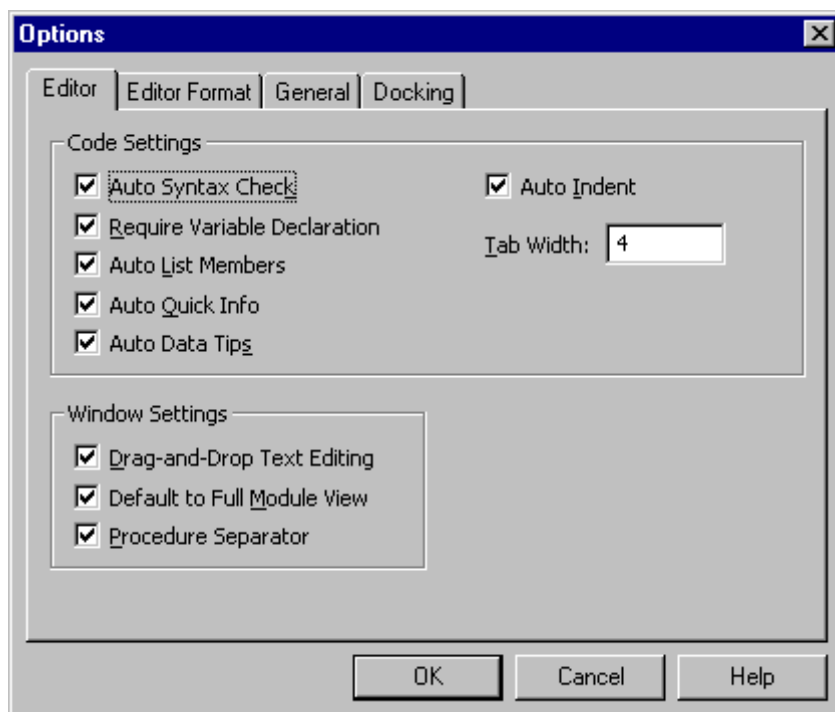


- In the **Code** view, the **Procedure** list box displays a list of all procedures in the module object:

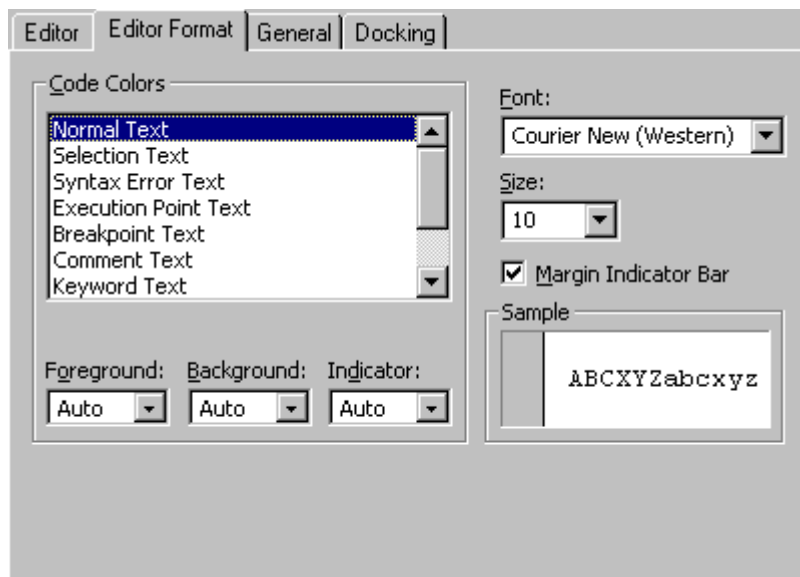


Setting the Visual Basic Editor Options

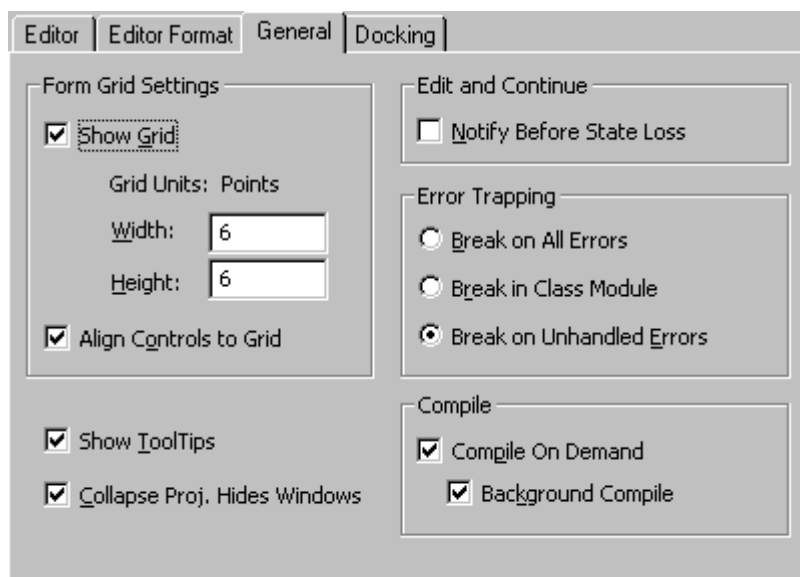
- Once you open the **Visual Basic Editor** window, you can change its features.
- From the main menu, choose **Tools > Options**.
- This opens up the **Options** dialog box:



- The **Code Settings** group controls the code editing.
- **Auto Syntax Check** option checks for syntax errors as you type the code.
- **Require Variable Declaration** option includes the Option Explicit declaration in new modules.
- **Auto List Members** option displays a list of keywords as you type the code.
- **Auto Quick Info** option displays syntax when you type a method or procedure name.
- **Auto Data Tips** option displays the current value of a variable when you rest your mouse pointer on the name.
- **Auto Indent** option indents a line of code to match the previous line.
- **Tab Width** option sets the number of characters for TAB key.
- The **Window Settings** group controls the editor window behaviours.
- **Drag-and-Drop Text Editing** option allows drag-and-drop editing in a module.
- **Default to Full Module view** option shows all procedures in the module by default.
- **Procedure Separator** option displays a horizontal line between procedures.
- Click the **Editor Format** tab to specify the appearance of the different types of text in the editor window:

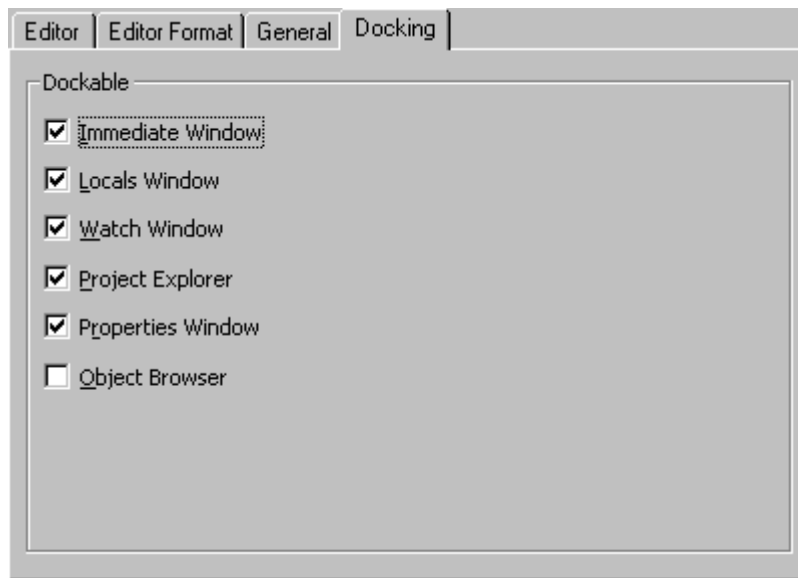


- Click the **General** tab to set Form Grid Settings, Error Trapping and Compile options:



SAMPLE

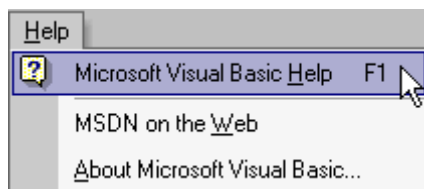
- Click the **Docking** tab to allow docking for editor windows:



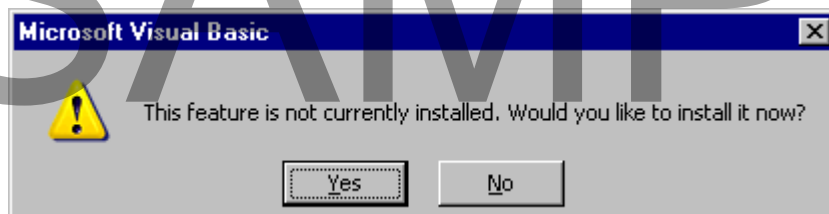
Getting Help with Visual Basic

Using Microsoft Visual Basic Help

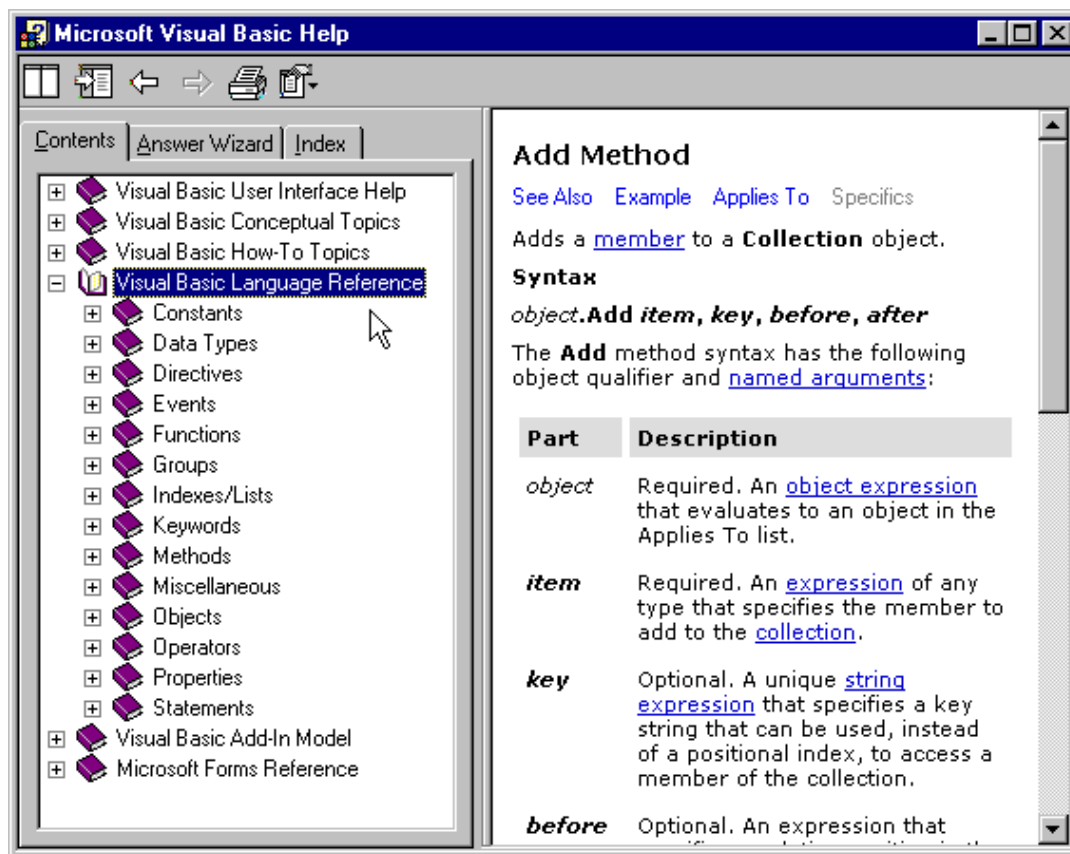
- Open the Visual Basic Editor window.
- From the main menu, choose **Help > Microsoft Visual Basic Help**
OR press the **F1** key
OR click on the **Microsoft Visual Basic Help** [?] button on the toolbar:



- If you get the following message, you must install the Microsoft Visual Basic Help first. Click on the **Yes** button, and follow the installation instructions:



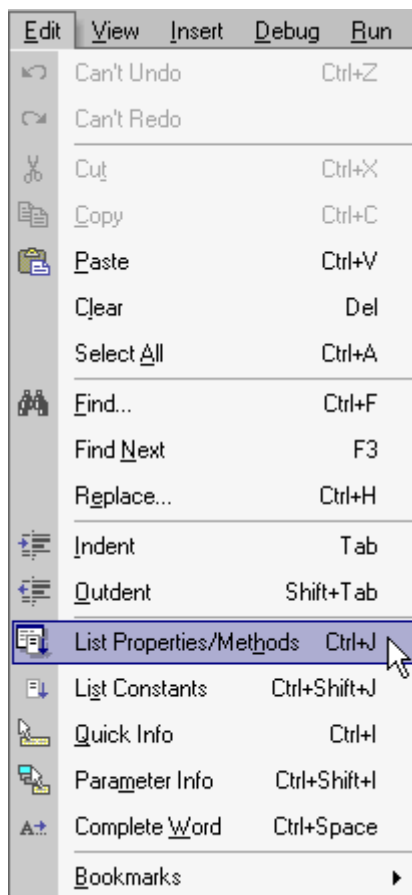
- The **Microsoft Visual Basic Help** window opens up. Here you can find extensive reference for various Visual Basic topics:



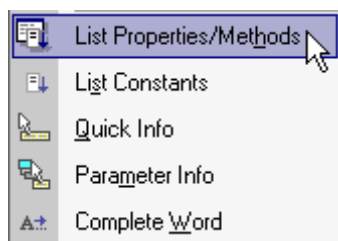
Getting Visual Basic Syntax Help

- While you are editing statements in the **Visual Basic Editor** window, you can get syntax help from the **Edit** menu:

SAMPLE



OR right-click inside the statement and choose desired help from the popup menu:



- **List Properties/Methods** lists all properties and methods available for selected statement:



- **List Constants** lists all intrinsic constants that are valid for a specific item in the argument list.

- **Quick Info** displays the complete syntax with the current item highlighted in bold:

```
CompanyNameFilters As OptionGroup
```

- **Parameter Info** displays the complete syntax with the current parameter highlighted in bold:

```
ApplyFilter([FilterName], [WhereCondition])
```

- **Complete Word** automatically completes the word when you begin typing.

Review Questions

How would you:

- Use Visual Basic Modules?
- Convert Macros to Visual Basic Code?
- Understand Modules?
- Create Modules?
- Understand Module Declarations?
- Understand Procedures?
- Use Naming Rules?
- Declare Variables?
- Set Variable Scope?
- Declare Constants?
- Use Methods?
- Use Arguments?
- Customize the Visual Basic Editor Window?
- Set the Visual Basic Editor Options?
- Use Microsoft Visual Basic Help?
- Get Visual Basic Syntax Help?

SAMPLE

End of the preview sample



This sample is approximately half of the full course. Please see the table of contents at the beginning of this document to see the full list of topics covered in the full course.

To purchase the rights to use the full training manuals at your training centre please see our web site at:

<http://www.cctglobal.com>

A courseware licence allows you to make unlimited copies for use at your training centre.

The IT Computer Courseware Library
A complete library of quality training courses

Includes Windows 7 and Office 2010 Courseware

- ▶ GET THE RIGHTS TO A COMPLETE LIBRARY OF TRAINING COURSES INCLUDING ALL THE MAJOR APPLICATIONS
- ▶ HIGH QUALITY, LOW COST COURSES
- ▶ ADD YOUR OWN NAME AND LOGOS
- ▶ PRINT AS MANY COPIES AS YOU NEED
- ▶ INTRANET VERSION ALSO AVAILABLE

The advertisement features a photograph of a diverse group of people, including a man in the foreground and several women behind him, all smiling.

In addition you get HTML formatted versions of each course, included with our printable courseware.

Invest in a complete Computer Courseware Library, including Windows 7 & Office 2010

The most cost effective courseware solution for your IT training needs. Get ALL our courses, and all new courses released within 12 months.

Over
7,000
Web Pages

Included when you purchase
the 'IT Courseware Library'.

SAMPLE