

# ExcelSQL Add-In

Version 3.1  
August 12th 2002

Copyright 2000-2002 by Noromaa Solutions Oy.  
All rights reserved.

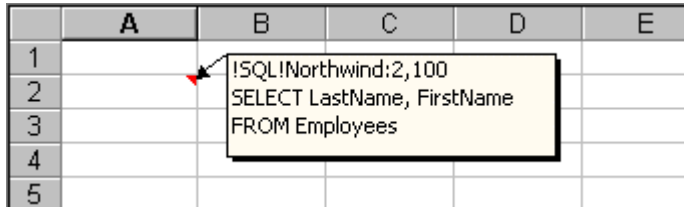
## Table of contents

1. Introduction .....	2
2. Installing ExcelSQL .....	3
3. Using ExcelSQL .....	4
Inserting Basic SQL Queries .....	4
ExcelSQL Query Components .....	4
Executing ExcelSQL Queries .....	6
4. References In ExcelSQL Queries .....	7
Using Absolute References .....	7
Using Relative References .....	8
More Complex References .....	9
Sharing Identical Queries Without Rewriting Them .....	10
5. Returning Special Data With Queries .....	11
Returning Cell Comments .....	11
Returning Line Breaks .....	12
Returning Formulas .....	12
6. Grouping Data .....	13
Grouping Definition Syntax .....	13
Grouping Definition Details .....	15
Advanced Grouping Example .....	15
7. Crosstabbing Data .....	18
Crosstab Definition Syntax .....	18
Result set considerations .....	19
8. Controlling Returned Rows And Columns .....	21
Returning Variable Numbers Of Rows .....	21
Copying Formulas To Inserted Rows .....	22
Transposing Output Data .....	23
9. Special Considerations With SQL Statements .....	24
Single Quotes In SQL References .....	24
Date Formats .....	25
Problems With Duplicate Column Names .....	26
Queries That Contain Several SELECTs .....	27
Queries That Return No Values .....	28
10. Miscellaneous Information .....	29
Useful Excel settings .....	29
Using ExcelSQL macros .....	29
11. ExcelSQL Cell Comment Syntax .....	30
Syntax Of The First Line .....	30
12. Troubleshooting .....	32
Information About Installation Problems .....	32
Common Installation Problems .....	33

# 1. Introduction

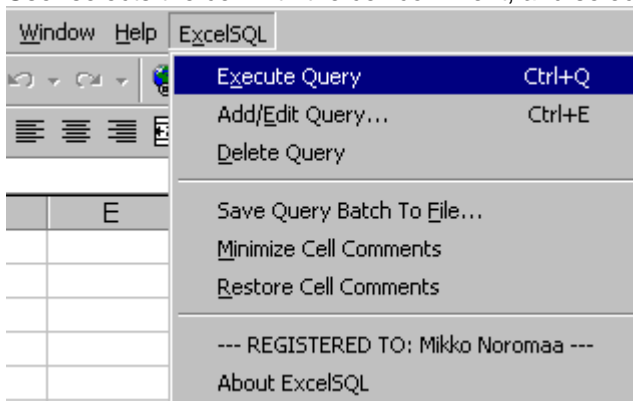
ExcelSQL is an add-in for Microsoft Excel. It can be used to fetch data from different SQL databases into Microsoft Excel. ExcelSQL works as follows:

1. User adds a cell comment into a cell in Excel. The comment contains a header recognized by ExcelSQL:



	A	B	C	D	E
1					
2		!SQL!Northwind:2,100 SELECT LastName, FirstName FROM Employees			
3					
4					
5					

2. User selects the cell with the cell comment, and selects Execute Query from ExcelSQL menu:



3. At this point, ExcelSQL reads the text from the cell comment, parses it, and sends it to the ODBC datasource defined in the cell comment (Northwind).
4. The database sends back results of the query. ExcelSQL processes the results, and writes them onto the Excel sheet starting from the cell where the comment was entered:



	A	B	C
1			
2		Davolio Nancy	
3		Fuller Andrew	
4		Leverling Janet	
5		Peacock Margaret	
6		Buchanan Steven	
7		Suyama Michael	
8		King Robert	
9		Callahan Laura	
10		Dodsworth Anne	
11			

## 2. Installing ExcelSQL

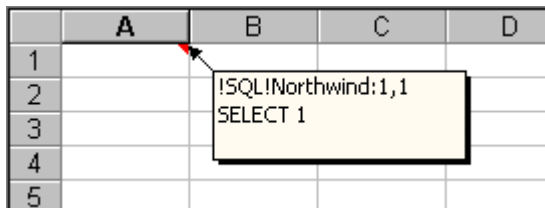
ExcelSQL works on Microsoft Excel 97, 2000 and XP. It requires that Data Access Objects (DAO) is installed.

To install ExcelSQL, do as follows:

1. Unzip the excelsql30.zip file to a directory on your computer (for example, C:\ExcelSQL). Extract folder structure as well.
2. Start Microsoft Excel, and select the Add-Ins command from the Tools menu.
3. In the Add-Ins dialog, click the Browse button. Select the ExcelSQL.xla file from your local directory (C:\ExcelSQL), either from the 97 or 2000 folder, depending on your Excel version. The 2000 version works on both Excel 2000 and Excel XP.

After installation, you should see the ExcelSQL menu appear. If you get any errors, see the Troubleshooting section at the end of this manual. The most common cause for unsuccessful installation is that Data Access Objects (DAO) is not installed, or a wrong version of it is installed. You can install DAO via the Microsoft Office Setup program.

Sometimes installation problems do not surface at the installation yet. It is strongly recommended that you test any new installation of ExcelSQL by executing a real query from a database. The query can be very simple, for example:



The image shows a screenshot of an Excel spreadsheet with columns A, B, C, and D, and rows 1 through 5. A yellow tooltip box is displayed over cell B2, containing the text: !SQL!Northwind:1,1  
SELECT 1. A red arrow points from the tooltip to cell B2.

Even if you don't have a datasource called Northwind defined on your computer, this query will test the most error-prone part of ExcelSQL, the link to DAO. This query should prompt for a username and password (to which you can enter anything), and then report an error about ODBC-connection failing. If this happens, you know that ExcelSQL works fine on your computer. If ExcelSQL reports another error, see the Troubleshooting section at the end of this manual.

### 3. Using ExcelSQL

#### *Inserting Basic SQL Queries*

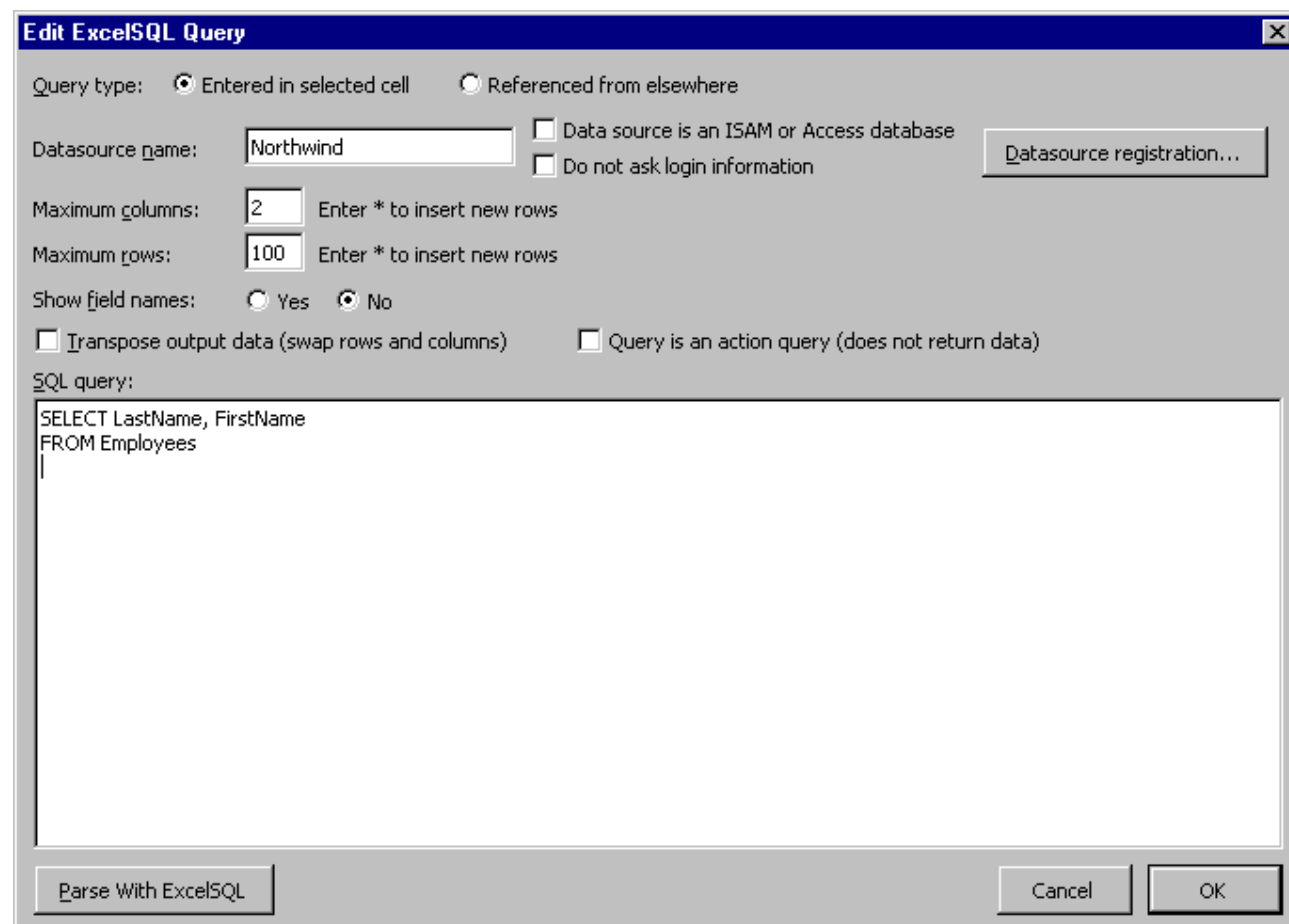
SQL queries executed by ExcelSQL are entered as cell comments in Excel. There are two ways to enter cell comments: 1) with Excel's own functionality (select Comment from the Insert menu), or 2) with ExcelSQL's query editor. The first option is most straightforward, and can be used on computers where ExcelSQL is not installed. The latter option is easiest, as the user does not have to remember the syntax of the ExcelSQL cell comment.

For details on entering ExcelSQL queries without the query editor, see the section ExcelSQL query syntax.

To start the query editor, select Add/Edit Query from the ExcelSQL menu.

#### *ExcelSQL Query Components*

The ExcelSQL query editor looks as follows:



The options in the ExcelSQL query editor are explained below:

- Query type** Either "Entered in selected cell" or "Referenced from elsewhere". For details about this option, see the section Sharing Identical Queries Without Rewriting Them later in this manual.
- Datasource name** This option specified the name of the datasource where the query will be sent. Datasources are defined in the ODBC Data Sources applet in Control Panel.
- Datasource is an ISAM or Access database**  
Select this checkbox if the you want to use an Access database (or another ISAM datasource) instead of an ODBC datasource. When this checkbox is checked, you must enter a filename with a fully qualified path into the **Datasource name** field.
- Do not ask login information**  
Select this checkbox if your datasource does not need login information (username and password). You must also select this checkbox if your datasource asks for login information in its own dialog box. For example, Lotus Notes datasources bypass ODBC login information and show their own dialog for asking a password.
- Datasource registration**  
If the datasource entered into the **Datasource name** does not exist, ExcelSQL can try to register it. To specify settings of how to register a datasource, click the **Datasource registration** button, and specify Driver, Server, Database, and other settings needed to register your database.
- Maximum columns** Specifies the maximum number of columns to return.
- Maximum rows** Specifies the maximum number of rows to return. If you don't know how many rows the query is going to return, enter an asterisk (\*) here. ExcelSQL will then insert space for new rows as they are returned.
- Show field names** If you select to show field names, ExcelSQL will write the column labels as part of its output. If you select No here, ExcelSQL will write only the data out.
- Transpose output** This setting specifies to "swap" rows with columns, and vice versa. For more information on the transpose setting, see the section "Transposing Output Data".
- Query is an action query**  
Select this checkbox if your query is not supposed to return any data, for example, if the query is an UPDATE or INSERT query. **Note:** If your query is a SELECT query that does not return any rows because of a WHERE criteria, do not select this checkbox! Even if the query doesn't return any rows, it still returns a valid resultset, even though it is an empty one.
- SQL query** This is where you enter your SQL query. Most of this manual describes the different options you can use in constructing the SQL query.

## ***Executing ExcelSQL Queries***

ExcelSQL queries are executed by selecting the Execute Query command from the ExcelSQL menu. You can also use the shortcut key, Ctrl+Q.

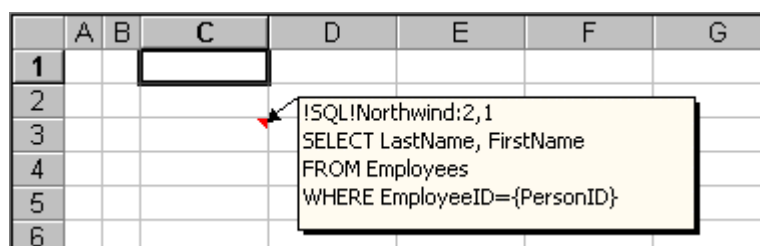
There can be two kinds of errors when executing a query: those that come from ExcelSQL and those that come from the SQL database. For example, if a reference cannot be parsed, the error is coming from ExcelSQL. On the other hand, if there is a comma missing from an SQL field list, it is an error coming from the SQL database.

## 4. References In ExcelSQL Queries

Because queries are stored in Excel cell comments, there is no Excel functionality for adding references to other cells. Since references are so essential in constructing any kind of an Excel solution, ExcelSQL includes the same features for adding references to cell comment as Excel does in using references in normal cells.

### Using Absolute References

All references are entered between curly brackets ( {}'s ). For example:



	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							

In this example, cell C1 is named PersonID. If cells are moved, rows are removed/added, or the ExcelSQL is placed in a different cell, the name still refers to the correct cell. This is why all absolute references used in ExcelSQL queries should use named cells. It is also the suggested practise to use in normal Excel formulas as it makes the formulas much more readable.

Another way to use absolute references is to use the A1-style notation:

**SELECT FirstName, LastName FROM Employees WHERE EmployeeID={C1}**

Note that in this case the reference is *absolute*. Normally, the reference "C1" in Excel is relative, and `$C$1` would be the absolute form. Relative reference is a reference which changes if it is copied to other cells. For example, if the reference C1 is copied from cell A1 to A2, it becomes C2 (pointing two columns right from the cell where the reference resides). Absolute reference stays always the same when copied between cells.

You can also use the normal absolute A1-syntax in ExcelSQL queries. In fact, A1, `$A$1`, `$A1` and `A$1` are all equal to ExcelSQL.

ExcelSQL will never change references when cell notes are copied, and thus absolute/relative distinction will not work for A1-style references. Note that if you add or remove rows/columns, Excel normally updates all formulas so that they refer to the same cells they originally referred to. For example, if there is a reference C1 in cell A1, and a new column is added between columns A and C, the reference is automatically updated to D1. ExcelSQL cannot do such updates, and all affected ExcelSQL query references must be manually changed.

**Hint:** ExcelSQL simply replaces all references written in {}'s before sending the query to the database server. If a referenced cell is empty, the resulting query could be, for example "...WHERE EmployeeID=". Of course, this will cause an SQL error to be returned from the server. You can prevent the error from occurring by preceding the reference with a 0: "...WHERE EmployeeID=0{PersonID}". In this case, the resulting query would be "...WHERE EmployeeID=0", which is syntactically correct, and would just return an empty recordset.

## Using Relative References

As absolute references, relative references are also entered between curly brackets ( {}'s ). To make the distinction between absolute and relative references, relative references include an equal sign before the reference. For example:

	A	B	C	D	E	F
1						
2		1				
3		2				
4		3				
5		4				
6						

```

!SQL!Northwind:2,1
SELECT LastName, FirstName
FROM Employees
WHERE EmployeeID={=RC[-1]}
    
```

Relative references must always be written in the R1C1-style, no matter what style is used in Excel formulas. In this example, "=RC[-1]" references the current row and the cell that is one cell left from the current column. You can easily see that you can copy this query to other cells, and it will work in them without modifications.

Standard R1C1-style constructs can be used in ExcelSQL references: Either row/column can be absolute, or even both. For example, "=RC1" references the column 1 of the current row. Thus, row is relative and column is absolute. A practical example is a matrix that can be filled with the same ExcelSQL query:

	A	B	C	D	E	F
1		<b>Year</b>				
2		<b>1996</b>	<b>1997</b>	<b>1998</b>	<b>1999</b>	
3	<b>1</b>					
4	<b>2</b>					
5	<b>3</b>					
6	<b>4</b>					
7	<b>5</b>					
8	<b>6</b>					
9						

```

!SQL!Northwind:1,1
SELECT COUNT(*)
FROM Orders
WHERE EmployeeID=0{=RC1}
AND YEAR(OrderDate)='{=R2C}'
    
```

This example shows the number of orders generated by an employee by year. Again, the same query can be used in all cells, and it will return the data for the correct employee and year.

## More Complex References

References can be nested inside each other. For example:

	A	B	C	D	E	F	G	H
1	Priority:	2						
2								
3	8							
4								
5								
6								
7	1	4,6	(Dairy Products, Meat/Poultry)					
8	2	5,7,8	(Grains/Cereals, Produce, Seafood)					
9	3	2,3	(Condiments, Confections)					
10	4	1	(Beverages)					
11								

!SQL!Northwind:2,\*  
 SELECT ProductID, ProductName  
 FROM Products  
 WHERE CategoryID IN (={R-1}{=RC[-1]}C2)

Here the innermost reference (=RC[-1]) is evaluated first. It gets the value from cell A3, which is the result of the MATCH formula. The outermost reference is then =R8C2, whose values is substituted into the SQL query. This system implements a simple "addition" to the SQL database by grouping product categories into 4 priority levels. Note that this kind of addition requires no changes to the SQL database structure.

After doing reference substitutions, ExcelSQL does **not** scan the substituted part for other references. Otherwise it would be completely impossible to use curly brackets in your SQL statements. If you want to create references based on reference substitutions, you must enclose the double-reference in two pairs of brackets. For example: "{ {ReferenceCell} }". If ReferenceCell contains the string "AnotherName", and AnotherName is a named cell on your worksheet, the reference will expanded to the contents of the AnotherName cell.

If you want to use curly brackets ( {}'s ) in your SQL statements, you must define two cells which contain the opening and closing bracket, and then reference these cells from your ExcelSQL query.

## Sharing Identical Queries Without Rewriting Them

There may be cases where you have long and complex queries which you want to use in several different cells. By using relative references, you can simplify your task so that all cells can use the same query (see the second example under section "Using relative references").

If the same long queries occupy hundreds of cells, the Excel workbook may grow enormously large in size (several megabytes). To reduce the filesize, you can write just one copy of your query and reference it from other cells. For example:

	A	B	C	D	E	F
1		Year				
2		1996	1997	1998	1999	
3	1					
4	2					
5	3					
6	4					
7	5					
8	6					
9						
10						
11						
12						
13						
14						
15						

Referencing cell notes this way is equivalent to typing the same query in the referencing cell note. Thus, the placement of the actual query (in this case, cell A10) has no effect. When the query in cell B4 is executed, all references are relative to cell B4, **not** cell A10.

You can also use A1-style references, and R1C1-style references (relative) as the query reference. The notes given under section "Using absolute references" applies here as well.

## 5. Returning Special Data With Queries

### Returning Cell Comments

Before writing query results out to the Excel sheet, ExcelSQL can process the results. For one, ExcelSQL write a specially formatted data as a cell comment. For example:

	A	B	C	D	E	F	G
1	ProductID	ProductName	UnitPrice	UnitsInStock			
2	1	Chai	\$18,00	39			
3	2	Chai	\$9,00	17			
4	24	Guar	\$4,50	20			
5	34	Sasq	\$4,00	111			
6	35	Stee	\$3,00	20			
7	38	Côte	\$3,50	17			
8	39	Chai	\$8,00	69			
9	43	Ipoh Coffee	\$46,00	17			
10	67	Laughing Lumberjack Lager	\$14,00	52			
11	70	Outback Lager	\$15,00	15			
12	75	Rhönbräu Klosterbier	\$7,75	125			
13	76	Lakkalikööri	\$18,00	57			
14							

The table shows a query result with columns ProductID, ProductName, UnitPrice, and UnitsInStock. A yellow callout box highlights the SQL query used: `SELECT ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder FROM Products WHERE CategoryID=1`. Another yellow callout box points to the value '10' in the UnitsInStock column for ProductID 39, with the text 'Units on order: 10'.

When the data returned from a query starts with a double equal sign (`==`), the value will be inserted as a cell comment in the **previous** result cell. Note that the width of the output is 4 in this example, even though the query has 5 columns.

You cannot return a cell comment for the first column because otherwise the ExcelSQL query cell comment might be lost.

Also note that if you return cell comments, you should be sure to return one for **every** row returned by the query. Otherwise some rows would be longer than others. Be especially careful if your query can contain a NULL value: in this case `'=='+NULL` equal NULL, and there would thus be no cell comment returned.

To remove an existing cell note, return `'=='` without any other text. To insert a line break into the formula, insert a line break in the ExcelSQL query.

If you want to return a real value beginning with two equal signs, prefix it with an apostrophe (`'`). This way ExcelSQL won't interpret it as a cell comment, and Excel will show it as text without the apostrophe.

**Note:** To create really advanced ExcelSQL solutions, you can return other ExcelSQL queries from your ExcelSQL queries.

## Returning Line Breaks

Sometimes you want the data on a single row of the resultset to be shown on several rows in Excel. For example, your query may return lots of fields and you must fit all of them into a single A4 page in portrait format. This is when you might want to take advantage of breaking the output to several lines.

When the data returned from a query contains three equal signs (===), ExcelSQL will wrap the rest of the fields to the next row. The three equal signs will **not** be written out to the Excel sheet. Be sure to return "===" for **every** row returned. Otherwise some rows will be longer than others.

## Returning Formulas

To return formulas from cell notes, prefix the formula with an equal sign (=). This is not really even ExcelSQL functionality, because Excel interprets all strings starting with an equal sign as formulas.

Following is an example of a query that returns formulas:

	A	B	C	D	E	F	G	H
1					2	List price		
2						Minimum customer price		
3						Average customer price		
4	1	Chai	14,4			Maximum customer price		
5	2	Chang	15,2					
6	3	Aniseed Syrn	8					
7	4	Chai						
8	5	Chai						
9	6	Grandma's						
10	7	Uncle Bob's						
11	8	Northwind						
12	9	Miscellaneous						
13	10	Ikura						
14	11	Quail						
15	12	Queso Manchego La Pastora	30,4					
16	13	Konbu	4,8					
17	14	Tofu	18,6					
18	15	Genen Shouyu	12,4					

Here cell D2 is named as "ShowVal", and the list in E1–E4 contains the choices in the dropdown box. When the dropdown box selection is changed, the value in cell D2 (ShowVal) changes from 1 to 4. The value shown in column C then changes depending on the selection.

Note that this query would have been better written to just return the 4 different values in different columns. However, sometimes kind of a selection possibility may be useful: for example, if you have lots of other formulas that calculate the data based on the returned products, maybe even drawing charts from that. Then you can simply change the "view" of the data from one point, and see how that affects the charts. This example also demonstrates a way to implement a 3-dimensional view in Excel.

Formulas returned from SQL queries must use the English language syntax even if you use a localized version of Excel. All functions must be in English (like IF), decimal separator must be a period (.), array separator must be comma (,) and semicolon (;), etc.

## 6. Grouping Data

Normally you do all data grouping in an SQL statement with a GROUP BY clause. In some cases, however, it is beneficial to do grouping (or part of grouping) in Excel. For example:

	A	B	C	D	E
1	CategoryName	ProductName	Orders	Quantity	
+	2	Beverages			
+	15	Condiments			
+	28	Confections			
+	42	Dairy Products			
+	53	Grains/Cereals			
+	61	Meat/Poultry			
-	68	Produce			
.	69	Longlife Tofu	13	297	
.	70	Manjimup Dried Apples	39	886	
.	71	Rössle Sauerkraut	33	640	
.	72	Tofu			
.	73	Uncle Bo			
+	74	Seafood			
	87				
	88				
	89				
	90				
	91				
	92				

```

ISQL!Northwind:10,100,1
SELECT C.CategoryName, P.ProductName, COUNT(*) AS Orders,
SUM(Quantity) AS Quantity
FROM Categories C JOIN Products P ON
P.CategoryID=C.CategoryID
JOIN [Order Details] OD ON OD.ProductID=P.ProductID
GROUP BY P.ProductID, C.CategoryName, P.ProductName
ORDER BY C.CategoryName, P.ProductName
{EXCELGROUP: _GROUP !OUTLINE}
    
```

This example shows products grouped by category, and the number of orders for each product. Note that part of the grouping was done in SQL (grouping by products), but a higher-level grouping by categories was done in Excel by ExcelSQL.

### Grouping Definition Syntax

To group data in Excel, a definition with the following syntax is added to some point in the ExcelSQL query:

```
{EXCELGROUP[:] _GROUP[SAME] [(!)OUTLINE];...}
```

The `_GROUP` keyword specifies that you are defining a column by which to group. `_GROUP` must always be the first EXCELGROUP definition. The `OUTLINE` specifier specifies to add an Excel outline level for this group. If you precede the `OUTLINE` specifier with an exclamation mark (!), the level will not be visible after the query is run (the plus buttons for expanding groups are shown instead).

If the EXCELGROUP definition does not specify the `_GROUP` keyword, it is considered a summary formula. Each group is shown on its own row (data is always on separate rows), so the row containing the group name can contain formulas for summarizing the data appearing below it. These formulas are entered as EXCELGROUP definitions. They can be specified either on separate EXCELGROUP items or on one EXCELGROUP item. In the latter case, separate keywords and summary formulas by semicolons (;). For example, the following definitions are identical:

```
{EXCELGROUP: _GROUP;=SUM({});=AVERAGE({) }
```

and

```
{EXCELGROUP: _GROUP}
{EXCELGROUP: =SUM({) }
{EXCELGROUP: =AVERAGE({) }
```

Both of these define a three-column query so that its first column will be used to group the results, and for each group, a sum of the second column values and an average of the third column values is calculated.

The `{}` character-combination used in the summary formula will be substituted by the range of the cells which the data of this grouping occupies. You can also enter fixed values and/or use cell references.

The following example enhances the products by category query of the first example by adding group totals for categories:

	A	B	C	D	E
1	CategoryName	ProductName	Orders	Quantity	
2	Beverages		404	9532	
15	Condiments		216	5298	
28	Confections		334	7906	
42	Dairy Products		366	9149	
53	Grains/Cereals		196	4562	
61	Meat/Poultry		173	4199	
68	Produce		136	2990	
69		Longlife Tofu	13	297	
70		Manjimup Dried Apples	39	886	
71		Rössle Sauerkraut	33	640	
72		Tofu			
73		Uncle Bo			
74	Seafood				
87					
88					
89					
90					
91					
92					

This way the results are meaningful even if the second level groups are never expanded. Thus the result is a true "drill-down" view where the user has a top-level view which can be drilled down for more details. An example with four different grouping levels is given in section "Advanced Grouping Example".

## Grouping Definition Details

You can define several grouping levels by specifying the `_GROUP` keyword more than once. However, there can only be one `_GROUP` keyword per `EXCELGROUP` item. To specify more groups, you must place each group in its own `EXCELGROUP` item.

Normally all groups are on their own row. All further data is placed on rows beneath the grouped row. If you want to place some groups on the same row as the previous group, use the `_GROUPSAME` specifier instead of the `_GROUP` specifier on the first group. `_GROUPSAME` specifies to place the **next** group on the same row as the group defined with `_GROUPSAME`. You can't use any summary columns for groups defined with the `_GROUPSAME` specifier, because there is no "spare" space on the group row.

The result set must be ordered by all grouping columns in the same order. ExcelSQL processes the results row by row, and compares each row to the preceding row. If the values are equal, the second row is assumed to be in the same group with the first one. If the values are different, a new group is started.

NOTE: Even though `EXCELGROUP` definitions are typically written into an SQL `SELECT` clause (after the grouped columns), the position of the `EXCELGROUP` definitions has no meaning at all. You could have the `EXCELGROUP` definitions before your query, after your query or split in several pieces anywhere inside the query text.

## Advanced Grouping Example

Following is an example with four grouping levels and some advanced grouping formulas:

```
ISQL!Northwind:8,*,1
SELECT C.CategoryName, P.ProductName, E.FirstName+' '+E.LastName AS Employee, Cu.CompanyName,
COUNT(*) AS Orders, SUM(OD.Quantity) AS Quantity, SUM(OD.UnitPrice*OD.Quantity) AS TotalPrice
FROM Categories C JOIN Products P ON P.CategoryID=C.CategoryID
JOIN [Order Details] OD ON OD.ProductID=P.ProductID
JOIN Orders O ON O.OrderID=OD.OrderID
JOIN Customers Cu ON Cu.CustomerID=O.CustomerID
JOIN Employees E ON E.EmployeeID=O.EmployeeID
GROUP BY P.ProductID, C.CategoryName, P.ProductName, Cu.CustomerID, E.EmployeeID,
E.FirstName+' '+E.LastName, Cu.CompanyName
ORDER BY C.CategoryName, P.ProductName, E.FirstName+' '+E.LastName, Cu.CompanyName

{EXCELGROUP: _GROUP !OUTLINE; =COUNTA({}); ; ; =SUM({})/3; =SUM({})/3; =SUM({})/3}
{EXCELGROUP: _GROUP !OUTLINE; ; ; =SUM({})/2; =SUM({})/2; =SUM({})/2}
{EXCELGROUP: _GROUP !OUTLINE; ; =SUM({}); =SUM({}); =SUM({}); =AVERAGE(OFFSET({},0,-1))}
{EXCELGROUP: _GROUP !OUTLINE}
```

The results of this query are as follows:

E3998					=SUM(E3999:E4049)/2							
1	2	3	4	5	A	B	C	D	E	F	G	H
	1	CategoryName	ProductName	Employee	CompanyN	Orders	Quantity	TotalPrice				
+	2	Beverages		12		404	9532	\$286 526,95				
+	898	Condiments		12		216	5298	\$113 694,75				
+	1415	Confections		13		334	7906	\$177 099,10				
+	2177	Dairy Products		10		366	9149	\$251 330,50				
+	2978	Grains/Cereals		7		196	4562	\$100 726,80				
+	3417	Meat/Poultry		6		173	4199	\$178 188,80				
-	3800	Produce		5		136	2990	\$105 268,60				
+	3801		Longlife Tofu			13	297	\$2 566,00				
+	3836		Manjimup Dried Apples			39	886	\$44 742,60				
+	3923		Rössle Sauerkraut			33	640	\$26 865,60				
-	3998		Tofu			22	404	\$8 630,40				
+	3999			Anne Dodsworth		1	5	\$116,25			\$116,25	
-	4002			Janet Leverling		3	30	\$571,95			\$190,65	
-	4003				Ana Trujillo Emparedados y helados							
-	4004					1	3	\$69,75				
-	4005				Hungry Coyote Import Store							
-	4006					1	15	\$279,00				
-	4007				Océano Atlántico Ltda.							
-	4008					1	12	\$223,20				
+	4009			Laura Callahan		3	52	\$1 116,00			\$372,00	
+	4016			Margaret Peacock		3	70	\$1 464,75			\$488,25	
+	4023			Michael Suyama		5	127	\$2 910,90			\$582,18	
+	4034			Nancy Davolio		4	68	\$1 343,85			\$335,96	
+	4043			Steven Buchanan		3	52	\$1 106,70			\$368,90	
+	4050		Uncle Bob's Organic Dried Pears			29	763	\$22 464,00				
+	4113	Seafood		12		330	7681	\$141 623,09				
	4855											

This query shows a deep drill-down view of orders in the Northwind database. The view above shows products in the "Produce" category, and we are examining Tofu sales. We can see that Tofu was sold to 3 companies by Janet Leverling, for a total of \$571.95 with an average order size of \$190.65.

Notes about the formulas used in the EXCELGROUP definitions:

- Formula in cell B3800 is "`=COUNTA(B3801:B4112)`". It is the count of products under the "Produce" category.
- Formula in cell E4002 is "`=SUM(E4003:E4008)`". This is self-explanatory. Same goes for formulas in cells F4002 and G4002. They are simple SUM formulas of the rows below them.
- Formula in cell H4002 is "`=AVERAGE(OFFSET(H4003:H4008;0;-1))`". The OFFSET function is required in this formula because {} is always substituted by a reference to the same column that the formula occupies. In this case, the average value is calculated from total prices that were in the previous column.
- Formula in cell E3998 is "`=SUM(E3999:E4049)/2`". The sum must be divided by two because there are other SUM formulas below this grouping level. The total sum thus includes the actual values plus the other SUM's. To get rid of the other SUM's, we divide the total sum by two. Same goes for formulas in cells F3998 and G3998.
- Formula in cell E3800 is "`=SUM(E3801:E4112)/3`". See the explanation above. This time there are actual values and two other SUM's in the range E3801:E4112. Thus we divide by three. Same goes for formulas in cells F3800 and G3800.
- The average formulas in column H are calculated only for the last grouping level. This is because on higher levels they would have included the SUM formulas in the TotalPrice column. AVERAGE cannot be fixed by a simple division the same way as the SUM formulas can be fixed. To calculate AVERAGE's on every grouping level, the SUM formulas must be placed in different columns than the actual data. In this example, the SUM formulas could be placed in column H and the AVERAGE formulas could then be in column I. This could be accomplished by adding one semicolon (;) to all EXCELGROUP definitions in the sample query.

A natural extension of the previous example is to make the fields shown in columns A, B, C and D variable. That is, the user could select the order in which the fields are shown. For example, the first column could be Employee if the user is interested in the sales totals broken down by employee rather than by product category. This can be done by using cell references inside the SELECT statement and by writing Excel formulas that return the desired SQL for each column in the desired order. This is left as an exercise to the reader.

## 7. Crosstabbing Data

Crosstab is a format where there are fields both on the Y axis and on the X axis. For example, following is an example of a crosstab:

	A	B	C	D	E
1	CompanyName	Federal Shipping	Speedy Express	United Package	
2	Alfreds Futterkiste	1	4	1	
3	Ana Trujillo Emparedados y helados	3	1		
4	Antonio Moreno Taquería	2	2	3	
5	Around the Horn	4	1	8	
6	Berglunds snabbköp	4	5	9	
7	Blauer See Delikatessen	3	1	3	
8	Blondesddsl père et fils	<pre>ISQL!Northwind:10,100,1 SELECT C.CompanyName, S.CompanyName AS Shipper, COUNT(*) AS Amount FROM Customers C JOIN Orders O ON O.CustomerID=C.CustomerID JOIN Shippers S ON S.ShipperID=O.ShipVia GROUP BY C.CustomerID, C.CompanyName, S.ShipperID, S.CompanyName ORDER BY C.CompanyName, S.CompanyName</pre>			
9	Bólido Comidas preparadas				
10	Bon app'				
11	Bottom-Dollar Markets				
12	B's Beverages				
13	Cactus Comidas para llevar				
14	Centro comercial Moctezuma				
15	Chop-suey Chinese				
16	Comércio Mineiro	{EXCELGROUP: _GROUPSAME}			
		{EXCELGROUP: _CROSSTABCOL}			
		{EXCELGROUP: _CROSSTABDATA}			

Getting this kind of an output by using SQL would require a major effort involving temporary tables and perhaps even UPDATE's in cursor fetch loops. With ExcelSQL, however, getting this kind of output is simple.

### Crosstab Definition Syntax

To create a crosstab output, use the `_CROSSTABCOL` and `_CROSSTABDATA` keywords inside the `EXCELGROUP` definition. There must be one `_GROUPSAME` (or `_GROUP`) specification and both of these `CROSSTAB` specifiers (only one of both). The `_GROUP` specification defines the rows in the crosstab (there can be more than one), `_CROSSTABCOL` defines the column in the crosstab, and `_CROSSTABDATA` defines the data shown in the "intersections". The two `CROSSTAB` specifiers must specify the last columns of the returned results.

With crosstab reports, you should most often use the `_GROUPSAME` specifier instead of the `_GROUP` specifier. With the `_GROUP` specifier, your crosstab data would appear on separate rows than the row heading data. This would usually make the crosstab look confusing. You can, however, freely mix the grouping specifiers with crosstabbing specifiers.

## Result set considerations

As with normal grouping (see section "Grouping Data"), the result set must be ordered by the grouping columns. The column headers are written out in the order they appear, so the result set must be ordered also by the crosstab column field. Note that if there are no rows for all columns' data, some column headings are written out later in the process. This may be a problem if you are using dates as crosstab columns. To prevent this, your SQL query should always return rows for all possible row/column combinations, even if the "intersection" data does not exist. The easiest way to accomplish this is to use outer joins and/or cross joins.

The following example illustrates the need to return rows for all columns.

	A	B	C	D	E	F	G	H	I	
1	Employee	1/1997	2/1997	3/1997	4/1997	5/1997	6/1997	7/1997	8/1997	
2	Steven Buchanan			2634,4		5127,5	3124,9	6475,4	3636,7	
3	Robert King	13703,4	3891	3867,2	5207,35	6041,25	2082	6144,6	7853,05	
4	Nancy Davolio	7331,6	2504,6	5493,9	240	9468,25	6112,65	19997,88	5119,1	
5	Michael Suy	SQLNorthwind:13,100,1						4228,3	1301	3982,25
6	Margaret Pe	SET NOCOUNT ON						4232,4	6104,8	16766,79
7	Laura Calla	-- Fill the temporary table #months with values from 1 to 12.						2616,05	3984,1	4756,5
8	Janet Leverl	DECLARE @year int SET @year=1997						5871,6	1109,15	5881,8
9	Anne Dods	CREATE TABLE #months(Month int)						3761,5	28	1928
10	Andrew Full	DECLARE @i int SET @i=1 WHILE @i<=12 BEGIN						7058,6	10320	57,5
11		INSERT INTO #months VALUES(@i)								
12		SET @i=@i+1								
13		END								
14		SELECT E.FirstName+' '+E.LastName AS Employee,								
15		''+CONVERT(varchar,m.Month)+''+CONVERT(varchar,@year),								
16		SUM(OD.UnitPrice*OD.Quantity) AS TotalAmount								
17		FROM (#months m CROSS JOIN Employees E)								
18		LEFT JOIN Orders O ON MONTH(O.OrderDate)=m.Month AND								
19		YEAR(O.OrderDate)=@year AND O.EmployeeID=E.EmployeeID								
20		LEFT JOIN [Order Details] OD ON OD.OrderID=O.OrderID								
21		GROUP BY E.EmployeeID, E.FirstName+' '+E.LastName, m.Month,								
22		''+CONVERT(varchar,m.Month)+''+CONVERT(varchar,@year)								
23		ORDER BY E.FirstName+' '+E.LastName DESC, m.Month								
24		DROP TABLE #months								
25		{EXCELGROUP: _GROUPSAME}								
26		{EXCELGROUP: _CROSSTABCOL}								

Without the temporary table and the CROSS JOIN, the rows for Steven Buchanen for January, February and April would have been missing. The first row would have been for March 1997. March 1997 would thus have been written to column B by ExcelSQL. The missing months (January, February and April) would have been added to new columns when the data of Robert King would have been processed. This means the columns would have been added after December (the picture is cut at August due to space constraints).

As you can see from the example above, returning rows for all possible columns can be quite tricky. Another way to ensure all columns are in their proper places is to return empty values for all columns in the beginning of the result set. This can be done with the SQL UNION operator as follows:

	A	B	C	D	E	F	G	H	I
1	Employee	1/1997	2/1997	3/1997	4/1997	5/1997	6/1997	7/1997	8/1997
2									
3	Steven Buchanan			2634,4		5127,5	3124,9	6475,4	3636,7
4	Robert King	13703,4	3891	3867,2	5707,35	6041,25	2082	6144,6	7853,05
5	Nancy Davolio	7331,6	2504,6	5493,9	240	9168,25	6112,65	19997,88	5119,1
6	Michael Suy	<pre> SQLNorthwind:13,100,1 SET NOCOUNT ON -- Fill the temporary table #months with values from 1 to 12. DECLARE @year int SET @year=1997 CREATE TABLE #months(Month int) DECLARE @i int SET @i=1 WHILE @i&lt;=12 BEGIN INSERT INTO #months VALUES(@i) SET @i=@i+1 END SELECT NULL AS Employee, ""+CONVERT(varchar,m.Month)+''+CONVERT(varchar,@year) AS MonthStr, NULL AS TotalAmount, m.Month INTO #data FROM #months m UNION SELECT E.FirstName+' '+E.LastName AS Employee, ""+CONVERT(varchar,MONTH(O.OrderDate))+''+CONVERT(varchar,YEAR(O.OrderDate)) AS MonthStr, SUM(OD.UnitPrice*OD.Quantity) AS TotalAmount, MONTH(O.OrderDate) AS Month FROM Employees E JOIN Orders O ON O.EmployeeID=E.EmployeeID JOIN [Order Details] OD ON OD.OrderID=O.OrderID WHERE YEAR(O.OrderDate)=@year GROUP BY E.EmployeeID, E.FirstName+' '+E.LastName, ""+CONVERT(varchar,MONTH(O.OrderDate))+''+CONVERT(varchar,YEAR(O.OrderDate)), YEAR(O.OrderDate), MONTH(O.OrderDate) SELECT Employee, MonthStr, TotalAmount FROM #data ORDER BY CASE WHEN Employee IS NULL THEN 'ZZZZZZZZZZ' ELSE Employee END DESC, Month DROP TABLE #months DROP TABLE #data {EXCELGROUP: _GROUPSAME} {EXCELGROUP: _CROSSTABCOL} {EXCELGROUP: _CROSSTABDATA} </pre>							
7	Margaret Pe								
8	Laura Callah								
9	Janet Leverl								
10	Anne Dodswo								
11	Andrew Full								
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									
26									
27									
28									
29									
30									
31									
32									
33									
34									

The FROM clause in this example is much simpler than that of the previous example. However, this example has the extra burden of making sure the returned months are sorted as number, not as text. If the results were returned straight from the two SELECT statements (combined with the UNION operator), the ORDER BY clause would have to reference the textual representation of the date. This would have resulted in a sorting order as follows: 1, 10, 11, 12, 2, 3, etc. By using a fourth column for the numeric month, we can sort the results properly. However, then we must get the SELECT output into a temporary table (#data) because we do not want to return the extra sorting column (Month) to ExcelSQL.

If the data can be sorted as text (i.e. it is something else than dates), sorting is not a problem, and this UNION method is much simpler than the CROSS JOIN method.

## 8. Controlling Returned Rows And Columns

### Returning Variable Numbers Of Rows

In the ExcelSQL query editor, you can specify the number of rows (and columns) your query returns. However, often the number of rows returned is not known beforehand. This may be a problem if you want to include some data below your query results. To circumvent this problem, you can use an asterisk (\*) as the number of rows to return. For example:

E3		=SUM(E1:E2)				
	A	B	C	D	E	F
1						
2						
3		<b>TOTAL</b>			0	
4		ISQL!Northwind:5,*,1 SELECT ProductID, ProductName, UnitPrice, UnitsInStock, '=RC[-2]*RC[-1]' AS StockValue FROM Products WHERE CategoryID=1				
5						
6						
7						
8						
9						
10						
11						

When this query is run, it will insert rows into the Excel sheet for each row returned by the query, except the first row. The result looks as follows:

E15		=SUM(E1:E14)				
	A	B	C	D	E	F
1	ProductID	ProductName	UnitPrice	UnitsInStock	StockValue	
2	1	Chai	18,00	39	702	
3	2	Chai	19,00	17	323	
4	24	Garden of Eatin'	4,50	20	90	
5	34	Spek	14,00	111	1554	
6	35	Spek	18,00	20	360	
7	38	Chocolate	63,50	17	4479,5	
8	39	Chocolate	18,00	69	1242	
9	43	Ispoh Coffee	46,00	17	782	
10	67	Laughing Lumberjack Lager	14,00	52	728	
11	70	Outback Lager	15,00	15	225	
12	75	Rhönbräu Klosterbier	7,75	125	968,75	
13	76	Lakkalikööri	18,00	57	1026	
14						
15		<b>TOTAL</b>			12480,25	
16						

When the query is run again, previously returned rows are deleted first. This way you can place sum formulas directly below the query results, as done in the example. Note, however, that you may not place any other data on the same rows with the query (that is, alongside with the query data), because that data would be deleted when the query rows are deleted.

When a query that returns variable numbers of rows is run, it will add a new named range onto your worksheet. This range has a name of the format: Z\_ExcelSQL\_cell where "cell" is the cell in which the query resides (Z is there to make it sort last in the list of names). If you move your query or delete this name, ExcelSQL will no longer be able to delete the correct rows before running the query. In this case, you must first delete any previous results manually.

If the query where the cell resides is named, the Z\_ExcelSQL range added by ExcelSQL will contain the name of the cell. For example, if the query cell is named as QueryCell, the range will be called Z\_ExcelSQL\_QueryCell. This means that if the query moves to another cell, ExcelSQL can still remove the correct rows when running the query. For this reason, it is highly recommended that cells containing queries are named.

### Copying Formulas To Inserted Rows

If the number of rows returned by a query is variable (see previous section, "Returning Variable Number Of Rows"), ExcelSQL deletes all previous rows that contain data. Sometimes, however, you want to copy formulas onto the rows that are inserted by ExcelSQL. For example, another way to implement the list in previous example is as follows:

	A	B	C	D	E	F
1	ProductID	ProductName	UnitPrice	UnitsInStock	StockValue	
2	1	Chef	\$18,00	39	702	
3	2	Chateau	\$19,00	17	323	
4	24	Garden of Eatin'	\$4,50	20	90	
5	34	Sausalito	\$14,00	111	1554	
6	35	St. Hubert	\$18,00	20	360	
7	38	Côte de Blaye	\$263,50	17	4479,5	
8	39	Chartreuse verte	\$18,00	69	1242	
9	43	Ipoh Coffee	\$46,00	17	782	
10	67	Laughing Lumberjack Lager	\$14,00	52	728	
11	70	Outback Lager	\$15,00	15	225	
12	75	Rhönbräu Klosterbier	\$7,75	125	968,75	
13	76	Lakkalikööri	\$18,00	57	1026	
14						
15		<b>TOTAL</b>			12480,25	
16						

The query in this example returns only the data, no formulas as the query in the previous example did. Instead, the StockValue formula is entered into column E. Note that the number of columns returned by the query has been set to 4 because otherwise the formulas in column E would be erased.

When ExcelSQL executes the query in this example, it first deleted all previous data. This situation is depicted below (row 2 is outside of the query result area):

	A	B	C	D	E	F
1	Prod	ProductName	UnitPrice	UnitsInStock	StockValue	
2						
3		TOTAL			0	
4						

After this ExcelSQL starts inserting new rows. When it inserts rows, it checks the next column after the last column returned by the query (column E in this case) for any formulas. If that cell contains formulas, the formulas are copied down to the newly inserted row. See the next picture for an illustration of how this process works:

	A	B	C	D	E	F	G	H
1	Prod	ProductName	UnitPrice	UnitsInStock	StockValue			
2	1	Chai	\$18.00	39				
3								
4		TOTAL			0			
5								

← Inserted row  
 ← Is there a formula here?  
 ← If yes, copy it here

It can be easily seen that after deleting results of the previous query, only one row with the StockValue formula remains: the heading row containing the field names. Thus the StockValue formula must be written so that it works also on the heading row. In this example, the formula checks on which row it is and either returns a heading ("StockValue"), or the actual result of the stock value calculation. Formulas may get quite complex this way, so the recommended way is to return the formulas from the query as shown by the example in section "Returning Variable Numbers Of Rows".

## Transposing Output Data

Normally, columns returned from the database become columns in Excel, and rows returned from the database become rows in Excel. Sometimes, however, you may want to reverse this so that columns become rows, and rows become columns. For example:

	A	B	C	D	E
1	Federal Shipping	Speedy Express	United Package		
2	255	249	326		
3					
4					
5					
6					
7					
8					
9					

```

ISQL!Northwind:10,2 !TRANSPOSE!
SELECT S.CompanyName, COUNT(*) AS Orders
FROM Shippers S
JOIN Orders O ON O.ShipVia=S.ShipperID
GROUP BY S.CompanyName
    
```

Note that Maximum columns and Maximum rows value are expressed in rows and columns in Excel. Thus, when you toggle the Transpose output data setting, you should swap the Maximum columns and Maximum rows values.

When the Transpose mode is on, all dimensions are reversed. For example, if you specify an asterisk (\*) as the number of columns, new columns are added as needed (normally rows are added, see section "Returning Variable Numbers Of Rows").

Some ExcelSQL functionality is not available when the Transpose mode is on. Grouping and crosstabbing is not available in Transpose mode. For more information on grouping and crosstabbing, see sections "Grouping Data" and "Crosstabbing Data".

## 9. Special Considerations With SQL Statements

### Single Quotes In SQL References

If the cells you reference contain single quotes, they will most probably be mis-interpreted in the SQL text. Take for example the following query:

	A	B	C	D	E	F
1	Company:	La maison d'Asie				
2						
3	CustomerID	ContactName	ContactTitle	Address	City	
4	LAMAI	Annette Roulet	Sales Manager	1 rue Alsace-Lorraine	Toulouse	
5		!SQL!Northwind:20,2,1				
6		SELECT CustomerID, ContactName,				
7		ContactTitle, Address, City				
8		FROM Customers				
9		WHERE CompanyName='{CompanyName}'				
10						

When the CompanyName reference is substituted into the ExcelSQL query, it would look as follows:

```
... WHERE CompanyName='La maison d'Asie'
```

Clearly, this would cause an SQL error, because there are non-matched quotes. However, ExcelSQL will correct this situation, and duplicate the single quote in the referenced value:

```
... WHERE CompanyName='La maison d''Asie'
```

ExcelSQL always substituted single quotes, if the reference is enclosed in single quotes. This automatic substitution works only if the single quotes are **immediately** before and after the reference (opening and closing curly brackets). If this is not the case, you will have to take care of quote substitution yourself. For example, Excel's SUBSTITUTE worksheet function could be used for this.

Note that if you use double quotes in your SQL statements (which is not ANSI-compatible anyway), automatic substitution will not happen.

## Date Formats

Always remember that values are passed to the SQL database as they appear on the worksheet. In some cases, however, the appearance can change from one computer to another, for example, date formats change depending on the computer's international settings. Thus, special care must be taken when dates are referenced from ExcelSQL queries.

Following is one possible solution of how to handle dates:

	A	B	C	D	E	F	G
1	Hire date:	31-1-1993	1/31/1993				
2							
3							
4							
5							
6							
7							
8							

!SQL!Northwind:4,100  
SELECT EmployeeID, LastName,  
FirstName, HireDate  
FROM Employees  
WHERE HireDate<='{HireDate}'

Here the date (entered in ANSI format in cell B1) is explicitly converted to the mm/dd/yyyy format.

## Problems With Duplicate Column Names

Sometimes ExcelSQL fails to return all columns specified in the query. For example:

	A	B	C	D
1	CompanyName	Country	Phone	
2	Around the Horn	UK	(503) 555-9831	
3	Ottilies Käselader		(503) 555-3199	
4	Folk och få HB		9831	
5	Lonesome Pine R		3199	
6	Wolski Zajazd		9831	
7	GROSELLA-Rest		3199	
8	Cactus Comidas		9931	
9	Piccolo und mehr		9831	
10	Bon app'	France	(503) 555-3199	
11	Chop-suey Chinese	Switzerland	(503) 555-3199	

MSQL!Northwind:5,100,1

```

SELECT C.CompanyName, C.Country,
S.CompanyName, S.Phone
FROM Customers C
JOIN Orders O ON C.CustomerID=C.CustomerID
JOIN Shippers S ON S.ShipperID=O.ShipVia
GROUP BY C.CompanyName, C.Country,
S.CompanyName, S.Phone

```

This query is supposed to return the companies and which shippers they have used. However, the CompanyName field is the the same in both tables, which causes ExcelSQL to process only the first one of them (this is a limitation in Data Access Objects, DAO).

You can make this query work by defining an alias for the other CompanyName column:

	A	B	C	D	E
1	CompanyName	Country	ShipperName	Phone	
2	Around the Horn	UK	Speedy Express	(503) 555-9831	
3	Ottilies Käselader		United Package	(503) 555-3199	
4	Folk och få HB		Speedy Express	(503) 555-9831	
5	Lonesome Pine R		United Package	(503) 555-3199	
6	Wolski Zajazd		Speedy Express	(503) 555-9831	
7	GROSELLA-Rest		United Package	(503) 555-3199	
8	Cactus Comidas		Shipping	(503) 555-9931	
9	Piccolo und mehr		Speedy Express	(503) 555-9831	
10	Bon app'	France	United Package	(503) 555-3199	
11	Chop-suey Chinese	Switzerland	United Package	(503) 555-3199	

MSQL!Northwind:5,100,1

```

SELECT C.CompanyName, C.Country,
S.CompanyName AS ShipperName, S.Phone
FROM Customers C
JOIN Orders O ON C.CustomerID=C.CustomerID
JOIN Shippers S ON S.ShipperID=O.ShipVia
GROUP BY C.CompanyName, C.Country,
S.CompanyName, S.Phone

```

## Queries That Contain Several SELECTs

ExcelSQL requires that all queries return a recordset. ExcelSQL cannot process several recordsets returned by a query; only the first recordset is processed.

If the first recordset is empty, ExcelSQL will report the error "Query did not return any rows". A common example is a variable assignment:

	A	B	C	D	E	F				
1										
2		<pre>!SQL!Northwind:5,100 DECLARE @category int SELECT @category=MAX(CategoryID) FROM Categories  SELECT ProductID, ProductName FROM Products WHERE CategoryID=@category</pre>								
3										
4										
5										
6										
7										
8										
9										

This query will report the error "Query did not return any rows", because ExcelSQL cannot process the results of the first query. The first query in this case is the SELECT which sets the @Category variable.

To make this query work, modify it as follows:

	A	B	C	D	E	F				
1										
2		<pre>!SQL!Northwind:5,100 SET NOCOUNT ON  DECLARE @category int SELECT @category=MAX(CategoryID) FROM Categories  SELECT ProductID, ProductName FROM Products WHERE CategoryID=@category</pre>								
3										
4										
5										
6										
7										
8										
9										
10										
11										

**Note:** The SET NOCOUNT ON clause is specific to Microsoft SQL Server. Other SQL products may work differently.

## Queries That Return No Values

Some perfectly valid SQL statements, for example UPDATES, do not return anything. As described in the section "Queries That Contain Several SELECTs" above, ExcelSQL only produces the first recordset returned. UPDATE queries return an empty recordset, and ExcelSQL will thus report the "Query did not return any rows" error. To suppress the error message, select the "Query is an action query" checkbox in the Query Editor (see the Using ExcelSQL section earlier in this manual).

Another possibility of suppressing the error message is to do a "dummy" select in addition to the UPDATE. This may be useful if you need to show an indication that the query was successful. For example:

	A	B	C	D	E	F	G
1		<b>New title</b>	<b>Update</b>				
2	1	Sales Representative	OK				
3	2	Vice President, Sales					
4	3	Sales Coordinator					
5	4	Sales Representative					
6							
7							
8							
9							

```
ISQL!Northwind:1,1
SET NOCOUNT ON

UPDATE Employees
SET Title='{=RC2}'
WHERE EmployeeID=0'{=RC1}

SELECT 'OK'
```

SET NOCOUNT ON causes the UPDATE clause not to return a recordset, and the last SELECT will thus be visible to ExcelSQL.

Note that a query whose WHERE conditions does not meet any records, is totally different from the case discussed here. For example:

	A	B	C	D	E
1					
2					
3					
4					
5					
6					

```
ISQL!Northwind:20,100
SELECT *
FROM Employees
WHERE EmployeeID=12345
```

This query will work perfectly, but it will not return anything.

## 10. Miscellaneous Information

### *Useful Excel settings*

To see a clear indication of cells that contain cell notes, you should have the cell note indicator on. This can be set from Options (in the Tools menu) under the View tab.

To make it easy to edit your SQL statement, turn In-Cell Editing off (Options, View tab). When In-Cell Editing is off, you can edit cell notes by double-clicking cells.

### *Using ExcelSQL macros*

If you want execute all SQL statements in a workbook in one go, run the macro "ExecuteAllSQLStatements". You can also add this macro as a button into your workbook. Note, that generally it is not a good idea to run all the SQL statements at once, because it may take a very long time.

When you select Execute Selected SQL Statements from the ExcelSQL menu, the macro "ExecuteSelectedSQLStatements" gets executed. You can run this macro from your own macros, for example to cause some queries to run automatically when your workbook is opened.

To use these ExcelSQL macros from your own macros, you must set a reference to Excelsql.xla first by selecting References from the Tools menu in the Visual Basic Editor.

# 11. ExcelSQL Cell Comment Syntax

## Syntax Of The First Line

The complete ExcelQL syntax for the first line in the cell note is defined as follows. Brackets ( [ ] ) mean optional values, vertical bars ( | ) separate interchangeable parts, and parenthesis ( ( ) ) are used to group related items. The vertical bar before the **connectstr** parameter is part of the syntax.

**!SQL(![\_]DSN[\$driver][|connectstr])[\_ISAMfile][=ref]:maxx,maxy[,headers] [!TRANSCOPE!]**

- !SQL** This is the string by which ExcelSQL recognizes the cell note as an SQL query. Regular cell notes are skipped, and not processed by ExcelSQL.
- DSN** Specifies the datasource for the query. You can specify any datasource that has previously been registered via the ODBC Administrator (32-bit). Prefix the datasource name with an underscore to specify that no login information should be asked from the user (in case the datasource asks for such information itself or just does not need a login). If you include an underscore, enclose the datasource definition in double quotes, for example: `"_MyDSN"`.
- driver and connectstr** You can register your own datasource by specifying **driver** and one or more connection string entries (**connstr**) separated by vertical bars ( | ). If the datasource is not registered, and you include **driver**, the datasource will be registered on the computer with the connection string you specify.
- ISAMfile** Specifies an ISAM datasource instead of an ODBC datasource. **ISAMfile** is the name of the file to connect to. It must be preceded by two underscores and enclosed in double quotes, for example: `"__C:\Files\Data.mdb"`.
- Following is an example of a Microsoft Access ExcelSQL specification:  
`!SQL!"__C:\Files\MyDatabase.mdb":...`
- ref** Specifies an absolute or a relative reference to a cell whose cell note will be used as the query to execute for this cell. If you use this argument, you may not specify any other settings (maxx, maxy or the SQL query) in the cell note.
- maxx** Specifies the maximum number of columns to return. Asterisk (\*) means that columns are to be inserted/deleted as needed (can only be used when the transpose mode is on).
- maxy** Specifies the maximum number of rows to return. Specify a very large value (16000, for example) if you want to allow returning as many rows as possible. That the area defined by **maxx** and **maxy** is cleared before executing the query. Asterisk (\*) means that rows are to be inserted/deleted as needed (can only be used when the transpose mode is off).
- headers** Specify 1 to insert column headers from the query results. Omit this parameters or specify 0 to not insert any headers.

**!TRANSPOSE!** If the SQL statement returns data in a "vertical" format but you would like to get it horizontally onto your worksheet, add this keyword to the cell note. Since the query will now return more columns than rows (typically), you must probably swap the **maxx** and **maxy** values.

**!ACTION!** Specifies that the SQL query is an action query that is not supposed to return any data, for example, if the query is an UPDATE or INSERT query.

## 12. Troubleshooting

### *Information About Installation Problems*

ExcelSQL relies heavily on the database functionality provided by Microsoft Data Access Objects (DAO). Because of different Excel and DAO versions, installing ExcelSQL may sometimes be tricky. This section describes the technical details behind these problems, and suggests solutions to the most common problems.

There are two versions of the ExcelSQL.xla file available: one for Excel 97 and one for Excel 2000/XP. The only difference between these versions is the version of DAO (Data Access Objects) they use. Excel 97 version uses DAO 3.5 and Excel 2000 version uses DAO 3.6. As long as the correct DAO library is installed on the computer, they can be used on either version of Excel, that is, ExcelSQL for 97 can be used on Excel 2000, and ExcelSQL for 2000 can be used on Excel 97.

For example, if you have upgraded Excel 97 to Excel 2000, you will have both versions of DAO installed. If you have not installed any Office 2000 software, you must use the 97 version of ExcelSQL. If you have not installed any Office 97 software, you must use the 2000 version of ExcelSQL.

There is an additional problem with Excel 97 and DAO: In typical installation of Office 97, DAO is **not** installed. Thus, ExcelSQL will not work on such an installation. You should ensure that all computer where ExcelSQL is to be used have DAO installed.

## ***Common Installation Problems***

### **Problem:**

"Can't find project or library" error is displayed, and Excel prompts for an ExcelSQLProject password when ExcelSQL is installed.

### **Description:**

The DAO version used by ExcelSQL is not found. Make sure you use the correct version of ExcelSQL (97 or 2000), and that DAO is installed. You can tell whether DAO is installed by looking at the directory C:\Program Files\Common Files\Microsoft Shared\DAO. DAO 3.5 (installed by Excel 97 and used by ExcelSQL for 97) is contained in file Dao350.dll, and DAO 3.6 (installed by Excel 2000 and used by ExcelSQL for 2000) is contained in file Dao360.dll.

### **Problem:**

"Compile error in hidden module: ExcelSQLModule" error is displayed.

### **Description:**

The DAO version used by ExcelSQL is not found. See the above problem for more information.

### **Problem:**

"ActiveX component can't create object" error is displayed when a query is run.

### **Description:**

The DAO library used by ExcelSQL is not found. The cause may be same as described in the first problem above, but it may also be caused by other causes. One such cause is that DAO license information is missing from the Windows Registry. To restore missing DAO license information, the license information can be entered into the Registry manually. For instructions on how to do this, see the Microsoft Knowledge Base article number Q240377. You can search for this article on the Microsoft Web site at <http://support.microsoft.com>.