

Contents

[Overview](#)

[Toolbar](#)

[Adding Tables to a Query](#)

[Add Table Dialog](#)

[Table Names in a Query](#)

[Selection Criteria for a Query](#)

[Query Options](#)

[Group Conditions in a Query](#)

[Sorting Query Results](#)

[Join Dialog](#)

[Options Dialog](#)

[Expression Dialog](#)

[SQL Window](#)

[Result Window](#)

Overview

The Visual Query Builder (VQB) is a powerful tool for building queries. You can use it to build queries visually, step-by-step. The advantage of building queries visually is that you can build and execute complex queries without knowledge of SQL. Even if you are an expert in SQL, since VQB works on different databases, you are insulated from learning the differences in their SQL syntax.

With VQB, you can build queries incrementally. This means that you can start with a simple query, execute it, see the results and refine it. You can repeat this process until you get the query you want. This process is known as 'drilling down' and is a common way of working with databases.











You can also use VQB as a tutorial for learning SQL. Since you can generate and display the SQL statements for the queries you create visually, you can understand SQL a lot faster by working with VQB.

With VQB, you can do the following:


- Select one or more tables to be used in the query.
For more information, refer to the topic [Adding Tables to a Query](#).
- Select columns to query using simple drag-and-drop.
- Reorder result columns or delete selected columns using drag-and-drop.
- Specify different types of join conditions between tables.
For more information, refer to the topic [Join Dialog](#).
- Generate an expression to be included in the SELECT list.
For more information, refer to the topic [Expression Dialog](#).
- Specify multiple selection criteria.
For more information, refer to the topic [Selection Criteria for a Query](#).
- Specify grouping and group criteria.
For more information, refer to the topic [Group Conditions in a Query](#).
- Specify how the results of a query can be sorted by one or more columns.
For more information, refer to the topic [Sorting Query Results](#).
- View the SQL statement for the query.
For more information, refer to the topic [SQL Window](#).
- Execute the query and browse the result.
For more information, refer to the topic [Result Window](#).
- Save the query in a file.

Toolbar

The Visual Query Builder includes a Toolbar that can be used to select the operations to be performed. The Toolbar icons and their functions are shown below.

Icon	Function
	Create a new query.
	Open an existing query. You will be prompted to choose the query file to open. The query files created by the Visual Query Builder have a .QRY extension.
	Save the query. This option saves the query under its original name. If no name was specified when the query was saved the first time, it is saved as UNTITLED.QRY.
	Save the query with a new name. You will be prompted to choose a file name. The extension .QRY is supplied by default.
	Set query options. For more information, refer to the topic Query Options .
	Select tables to query. For more information, refer to the topic Adding Tables to a Query .
	Build a result column as an expression. For more information, refer to the topic Expression Dialog .
	Show SQL statement. For more information, refer to the topic SQL Window .
	Run the query. For more information, refer to the topic Result Window .
	Exit Visual Query Builder. You will be prompted to save the current query if you made any changes.

Adding Tables to a Query

To add a table to a query, choose the **Add Table** icon  from the Toolbar. The Add Table dialog will be displayed.

The **Add Table** dialog lists the names of tables in the selected data source. You can add a table to the query either by double-clicking on the table name or by selecting the table name and then choosing the **Add** command button.

Table frames for each table included in the query are displayed in the Visual Query Builder workspace.


Add Table Dialog

[See Also Example](#)

You can add tables to a query using the **Add Table** dialog. Tables can be added when:

- You start building a query
- You modify an existing query.

The **Add Table** dialog is automatically displayed when you start the Visual Query Builder.

To display the **Add Table** dialog at any time, choose the **Add Table** icon  from the **Toolbar**.

The **Add Table** dialog lists the names of all tables in the current data source. If you want to see the system tables as well, you can check **Include System Tables**.

1. To add a table to the query, do one of the following:
 - Select the table name from the list of tables displayed and choose **Add**.
 - Double-click on the table name.
2. After adding the required tables, choose **Close**.



The tables selected for the query will now be displayed in the Visual Query Builder workspace.
3. To add a column from one of the selected tables to the query, do one of the following:
 - Select the column name from the list of column names displayed in the table frame. Drag the column and drop it on the query grid at the bottom of the screen.
 - Double-click on the column name.

The selected column will be displayed in the query grid.
4. You can create joins between the tables selected for the query. For more information, refer to the topic [Join Dialog](#).

Add a Table - Example

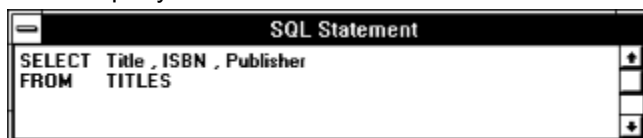
Query: List the title, ISBN and publisher id of each book (In all examples included in the help file, we assume that the tables and columns exist as described).

To perform this query, you must first select the **TITLES** table (because that is where you know the information exists) and then select the **Title** column, **ISBN** column and **Publisher** column. With VQB, you perform the following steps:

1. Make sure that the **Add Table** dialog is displayed and double-click the **TITLES** table name.
Alternatively, select the table name from the list of tables and choose the **Add** command button. A table frame is displayed in the query workspace.
2. Choose **Close**.
We need to select only one table for this query. The **TITLES** table frame will be displayed in the Visual Query Builder workspace with the column names in the table.
3. Double-click the **Title** column in the table frame.
This adds the column name to the query grid. The **Table Name** row for the column shows **TITLES**, signifying that the column belongs to the **TITLES** table. **Option** shows **Show**, signifying that the column's value will be displayed in the query result.
4. Select the **ISBN** column and drop it on the query grid. This is an alternative way of adding a column to a query. When you drag the column name across the query workspace, the mouse pointer changes to , signifying that the column cannot be dropped anywhere in the workspace. When you reach the query grid, the mouse pointer changes to , signifying that it can be dropped in that position.
If you drop the new column on a column that is already on the query grid, the new column name is inserted before the existing column. If you place the new column in the blank area to the right of the query grid, it is appended as the last column of the query.
To select all columns from a table, double-click the table caption. The column names will be highlighted. Hold the left mouse button down and drag the column names to the query grid.
5. Add the **Publisher** column to the query.
6. Choose **Run** from the Toolbar.
A result window is displayed with all records in the **TITLES** table with the field order you specified. You can scroll through the result set using the vertical scroll bar. When you are finished looking at the result set, double click on the system menu to close the result window and return to the Visual Query Builder main window.

If you want to remove duplicate records from the result, select **Option** from the Toolbar. In the **Options** dialog box displayed, check the **Remove Duplicate Records** check box and choose **OK**.

7. Choose **SQL** from the Toolbar.
The **SQL Statement** window is displayed as shown. It shows the SELECT statement for the query.



Note that the SELECT statement contains the three columns we selected. The SELECT statement in this case is very simple. It contains the SELECT keyword, a list of column names

and the FROM keyword followed by the name of the table. As you learn how to use the Visual Query Builder, you will see more complex SELECT statements.

The **SQL Statement** window is a non-modal window. You can, therefore, return to the Visual Query Builder without closing the window. You can size the window suitably and position it on the desktop so that the SQL statement is visible always. If you change the query or selection criteria, the SQL statement will be updated immediately. If you are new to SQL, this is a good way to learn SQL syntax.

8. Choose **Save As** from the Toolbar. In the **Save Query** dialog box, navigate to a directory where you want to save the query and save it with the name **titles**. The .QRY extension is automatically added by Visual Query Builder.

See Also

[Adding Tables to a Query](#)

[Table Names in a Query](#)

[Join Dialog](#)

Table Names in a Query

[See Also](#)

The **Table Name** row in the Visual Query Builder grid displays the name of the base table from which the query column is derived. If you have multiple tables included in a query, the table name corresponding to each column is displayed in the **Table Name** row. The column name itself is displayed right on top.

If you create an **Expression** column, the actual expression is displayed in the **Table Name** row. Each base column included in the expression is qualified by the table name in this case.

See Also

Expression Dialog

Join Dialog

Selection Criteria for a Query

Example 1

Example 2

You can specify the selection criteria for a query in the **Criteria** row of the query grid. All selection criteria allowed in the WHERE clause of a SQL SELECT statement are valid. This includes =, >, <, !=, LIKE, BETWEEN, and IN clauses. To make query selections that have to be ANDed together, you specify multiple conditions in the **Criteria** row. To specify selection criteria that have to be ORed, you specify the selection conditions in the **Criteria** row and **Or** row.

You can enter up to 255 characters in a cell within the **Criteria** row. This means that the selection criteria specified for a column should not exceed 255 characters.

Selection Criteria - Example 1

Query: List the title and ISBN of books whose titles include the pattern *SQL* (In all examples included in the help file, we assume that the tables and columns exist as described).

1. Select the **TITLES** table using the **Table** command.
2. Select the **Title** and **ISBN** columns to query.
3. In the query grid, bring the **Criteria** row into view using the vertical scroll bars.
4. Position the cursor in the **Title** column in the **Criteria** row.
5. Type **LIKE '%SQL%'** in the cell. Include the single quotes.
This selection specifies that books whose titles include the pattern **SQL** anywhere in the title should be listed.
6. Choose **Run**.
The result window displays the books whose titles contain the pattern **SQL**.
7. The SELECT statement for this query will be:

```
SELECT Title , ISBN  
FROM TITLES  
WHERE ( Title LIKE '%SQL%' )
```
8. Save the query with the name **sqlbooks**.

Selection Criteria - Example 2

Query: List the title, ISBN, publisher, price, and discount on books which are published by ADWE or which cost more than \$35 (In all examples included in the help file, we assume that the tables and columns exist as described).

1. Select the **TITLES** table using the **Table** command.
2. Select the **Title**, **ISBN** and **Publisher** columns to query.
3. In the query grid, bring the **Criteria** and **Or** rows into view using the vertical scroll bars.
4. Position the cursor in the **Publisher** column in the **Criteria** row.
5. Type **'ADWE'** in the cell. Include the single quotes.
6. Add the **Price** and **Discount** columns to the query.
7. In the **Or** row, type **>35** in the **Price** column.

The selection criteria specify that books published by publisher ADWE or books costing more than \$35 should be listed.

8. Run the query.
9. The SQL statement for this query will read as follows:

```
SELECT Title , ISBN , Publisher , Price , Discount  
FROM TITLES  
WHERE ( Publisher = 'ADWE' ) OR ( Price >35 )
```
10. Save the query as **pubprice**.

Query Options

The **Option** row in the Visual Query Builder grid is used for the following:

- To hide or show a query column.
For more information, refer to the topic [Hiding a Query Column](#).
- To specify column aggregates such as COUNT, SUM, MAX, MIN, and AVG.
For more information, refer to the topic [Specifying Aggregates](#).
- To specify grouping criteria.
For more information, refer to the topic [Specifying Grouping Criteria](#).

Hiding a Query Column

Example

While creating queries, you may want to select rows based on some selection criteria, but not want to display the value of the selection criteria in the query result. For example, you may want to display the last names and ids of all employees in the production department. We need the last name, id and department columns to frame this query. However, we need not display the department column, since all records in the query result will have the same value for that column.

To hide a query column, proceed as follows:

1. In the query grid, place the mouse pointer on the intersection of the column you want to hide and the **Option** row.
2. Click the mouse button.
A popup menu appears with the **Show** menu item checked.
3. Click the mouse button to uncheck the menu item.
The **Show** keyword will no longer be displayed in the **Option** row. The selected column will not be included in the query result. However, since the column is included in the query, you can still specify selection criteria.
4. The **Show** option works as a toggle. To re-display a hidden column, therefore, follow the above steps as they are. The **Show** popup menu item will now be checked and the **Show** keyword will be displayed in the **Option** row.

Hide a Query Column - Example

Query: List the title, ISBN and price of books on sale (In all examples included in the help file, we assume that the tables and columns exist as described).

This query is based on the **TITLES** table. The **OnSale** column in the table indicates whether a book is on sale. If the column contains the value **'T'**, the book is on sale; if it contains the value **'F'**, it is not. The **OnSale** column has **Character** data type and therefore, the value entered should be enclosed in single quotes (**'**).

The above query should, therefore, be phrased as:

Find the Title, ISBN and Price for Titles where OnSale = 'T'.

Described below are the steps for building this query.

1. Select the **TITLES** table using the **Table** command.
2. Select the **Title, ISBN, Price** and **OnSale** columns to query. Use the vertical scroll bars on the table frame to bring the **Price** and **OnSale** columns into view.
3. In the query grid, bring the **Criteria** row into view using the vertical scroll bars. If necessary, resize the VQB main window. We will not need the **Group Condition** or the associated **Or** criteria in this query.

The **Table Name, Option** and **Sort** rows remain in view when you scroll vertically. These rows display information that you may want to refer to always.

4. Position the cursor in the **Criteria** row and the **OnSale** column. You can change the active cell either using the mouse or using the arrow / TAB keys.
5. Type **'T'** in the cell. Include the single quotes.
6. Choose **Run**.

The result window displays only those titles which are on sale.

7. View the SELECT statement for the query. It includes the row selection criteria.

```
SELECT Title , ISBN , Price , OnSale
FROM   TITLES
WHERE  ( OnSale = 'T' )
```

8. Save the query with the name **onsale**.

All rows in this query result will contain the value **'T'** in the **OnSale** column. We need not, therefore, display that column. To remove the column from the SELECT list, proceed as follows:

9. Position the cursor in the **OnSale** column in the **Option** row.
10. Click the mouse button.
A popup menu appears with the **Show** menu item checked.
11. Click the mouse button to uncheck the menu item.
The **Show** keyword will no longer be displayed in the **Option** row.
12. Run the query.
The **OnSale** column does not appear in the query result.
13. The SQL statement for the query will now read as follows:

```
SELECT Title , ISBN , Price
```



```
FROM TITLES  
WHERE ( OnSale = 'T' )
```

14. Using the **Save** command, save the query under its original name.

Specifying Aggregates

[See Also](#)

[Example](#)

You can group column values by specifying a GROUP BY condition. To do so, position the cursor in the column to group in the **Option** row and click the mouse button. A popup menu appears. From the popup menu, select **Group**. The Visual Query Builder automatically assigns a group number for each column. If you remove a column from the query on which a group condition is defined, the GROUP BY condition on the column will be automatically deleted and the group numbers on other group columns automatically re-assigned.

You can specify aggregate operations for non-grouped columns. These include SUM, COUNT, AVG, MIN and MAX. Aggregate operations are also selected from the popup menu.

To define an aggregate of a numeric expression, you can do one of the following:

- a) Create the expression using the **Expression** dialog and then define the aggregate in the query grid.
- b) Define the expression and the aggregate directly in the **Expression** dialog.

Specifying Grouping Criteria

[See Also](#)

[Example](#)

You can group column values by specifying a GROUP BY condition. To do so, position the cursor in the column to group in the **Option** row and click the mouse button. A popup menu appears. From the popup menu, select **Group**. The Visual Query Builder automatically assigns a group number for each column. If you remove a column from the query on which a group condition is defined, the GROUP BY condition on the column will be automatically deleted and the group numbers on other group columns automatically re-assigned.

You can specify aggregate operations for non-grouped columns. These include SUM, COUNT, AVG, MIN and MAX. Aggregate operations are also selected from the popup menu.

To define an aggregate of a numeric expression, you can do one of the following:


- a) Create the expression using the **Expression** dialog and then define the aggregate in the query grid.
- b) Define the expression and the aggregate directly in the **Expression** dialog.

Grouping and Aggregates - Example

Query: List the number of orders and total value of orders for each customer (In all examples included in the help file, we assume that the tables and columns exist as described).

To answer this query, we need to join the **CUSTOMER**, **ORDERS** and **DETAILS** tables. To get the number of orders for each customer, we will use the COUNT aggregate function. To get the total value of orders for each customer, we will use the SUM aggregate function.

To build and test the query, perform the following steps:

1. Choose **New** to create a new query and choose the **Add Table** icon  from the Toolbar. The **Add Table** dialog box will be displayed.
2. Add the **CUSTOMER**, **ORDERS** and **DETAILS** tables to the query and close the **Add Table** dialog. For more information on how to add tables to a query, refer to the topic Add Table Dialog.
3. Define a join between the **CUSTOMER** and **ORDERS** tables using the **CustomerId** column. For more information on how to define joins, refer to the topic Join Dialog.
4. Define a join between the **ORDERS** and **DETAILS** tables using the **OrderId** column.
5. Add the **CustomerId**, **FirstName** and **LastName** columns from the **CUSTOMER** table to the query grid. Also add the **OrderId** column from the **ORDERS** table. For more information on how to add table columns to a query, refer to the topic Add Table Dialog.
6. Position the cursor in the **CustomerId** column in the **Option** row. Click the mouse button. From the popup menu displayed, choose **Group**.
This step signifies that the query results will be grouped by the **CustomerId** column. The **Option** row shows **Grp(1)**, indicating that the **CustomerId** column is the first GROUP BY column. If you have additional GROUP BY columns, they will be marked as **Grp(2)**, **Grp(3)**, ... etc.
7. In a similar way, define GROUP BY criteria for the **FirstName** and **LastName** columns.

Remember that SQL syntax requires a GROUP BY condition to include all columns not involved in an aggregate operation. In our example, we need to define three GROUP BY columns.

8. Position the cursor in the **OrderId** column in the **Option** row. Click the mouse button. From the popup menu displayed, choose **Count**.

This step signifies that the number of orders should be counted for each group.

To get the total value of orders for each customer, we need to define an expression. To do so, proceed as follows:

9. Choose **Expr** from the **Toolbar**.
The **Expression** dialog will be displayed.
10. Change **Expression Name** to **Total_Order_Value**.
11. Double-click **sum(X)** from the **Functions** list.
The **Expression** edit box will display **sum(X)** with **X** highlighted.
12. Select **DETAILS** from the **Tables** pulldown list.
The columns in the **DETAILS** table will be displayed in the **Columns** list.
13. Double-click the column name **Quantity**.

- The column name replaces **X** in the **Expression** edit box.
14. Double-click the multiplication operator (*) in the **Operators** list.
The multiplication operator is appended to the sum expression.
 15. Double-click the **SalePrice** column in the **Columns** list.
The column name is appended to the sum expression.
 16. Choose **Done**.
You will be prompted to save the expression definition. Choose **Yes**.
You will return to the VQB main window. The **Total_Order_Value** expression will be included as the last column in the query.
 17. Run the query.
The number of orders and total order value for each customer will be displayed.
 18. The SQL statement for this query will read as follows:

```
SELECT CUSTOMER.CustomerId , FirstName ,
        LastName , count( ORDERS.OrderId ) ,
        ( sum( DETAILS.Quantity * DETAILS.SalePrice ) ) as Total_Order_Value
FROM   CUSTOMER , DETAILS , ORDERS
WHERE  ( CUSTOMER.CustomerId = ORDERS.CustomerId )
        AND
        ( ORDERS.OrderId = DETAILS.OrderId )
GROUP BY CUSTOMER.CustomerId , FirstName , LastName
```
 19. Save the query as **totalord**.

See Also

Expression Dialog

Group Conditions in a Query

[See Also](#)

[Example](#)

In a query with GROUP BY conditions, you can perform selections on the aggregate columns. For example, from an order database, you may want to create a list of salespersons whose total order bookings for the year exceed \$1,000,000. This requires grouping the orders by salesperson and then selecting the salesperson records where the sum exceeds the specified target. To do so, we need to define a SUM aggregate for order value and then apply a selection criterion to the aggregate. This is the same as specifying a HAVING condition in a SQL SELECT statement with a GROUP BY clause.

The Visual Query Builder allows you to specify a HAVING condition on an aggregate column by typing in the selection criteria in the **Group Condition** row. The HAVING keyword is automatically added and need not, therefore, be specified.

Group Conditions (HAVING Clause) - Example

Query: List the number of orders and total order value for customers whose total orders exceed \$500 (In all examples included in the help file, we assume that the tables and columns exist as described).

To answer this query, we need to add a HAVING condition to query demonstrated in the topic [Grouping and Aggregates - Example](#). You can do this as follows:

1. Make sure that you have the **totalord** query in the Visual Query Builder workspace.
2. Position the cursor in the **Total_Order_Value** column in the **Group Condition** row.
3. Type **>500**.

This signifies that only those rows where the SUM of order value exceeds 500 should be listed.

4. Run the query.

You will see the list of customers whose total orders exceed \$500.

5. The SQL statement for this query will read as follows:

```
SELECT CUSTOMER.CustomerId , FirstName ,
        LastName , count( ORDERS.OrderId ) ,
        ( sum( DETAILS.Quantity * DETAILS.SalePrice ) ) as Total_Order_Value
FROM    CUSTOMER , DETAILS , ORDERS
WHERE   ( CUSTOMER.CustomerId = ORDERS.CustomerId )
        AND
        ( ORDERS.OrderId = DETAILS.OrderId )
GROUP BY
        CUSTOMER.CustomerId , FirstName , LastName
HAVING ( ( sum( DETAILS.Quantity * DETAILS.SalePrice ) ) >500 )
```

6. Save the query as **ordgt500**.

See Also

Specifying Grouping Criteria

Sorting Query Results

Example

You can sort query results in ascending or descending order of selected columns. To do so, position the cursor under the column name on the **Sort** row. Click the mouse button to get a popup menu. Select **Ascending** or **Descending** from the popup menu. You can mix the **Ascending** and **Descending** options in one query.

You can specify up to 8 sort columns in a query. The number of columns for sorting may, however, be dependent on the database management system and the ODBC driver used.

Sorting - Example

Query: *List customer names, number of orders and total order value for each customer in descending order of total order value* (In all examples included in the help file, we assume that the tables and columns exist as described).

Some ODBC drivers and / or databases do not allow ordering on calculated column values. This query will not work with those databases.

To answer this query, we need to make use of an `ORDER BY` condition. We will do this using the query demonstrated in the topic **Grouping and Aggregates - Example**.

1. Make sure that you have the ***totalord*** query in the Visual Query Builder workspace.
2. Position the cursor in the ***Total_Order_Value*** column in the **Sort** row.
3. Click the mouse button and select **Descending** from the popup menu displayed.
This signifies that the result should be arranged in descending order of total order value.
4. Run the query.

You will see the list of customers, number of orders and total order value in the reverse order of total order value.

5. The SQL statement for this query will read as follows:


```
SELECT CUSTOMER.CustomerId , FirstName , LastName , count( ORDERS.OrderId ) ,  
       ( sum( DETAILS.Quantity * DETAILS.SalePrice ) ) as Total_Order_Value  
FROM   CUSTOMER , DETAILS , ORDERS  
WHERE  ( CUSTOMER.CustomerId = ORDERS.CustomerId )  
       AND  
       ( ORDERS.OrderId = DETAILS.OrderId )  
GROUP BY CUSTOMER.CustomerId , FirstName , LastName  
ORDER BY 5 desc
```


6. Save the query as ***orddesc***.

Join Dialog

Example

In many cases, you have to combine information from more than one table to perform a query. For example, you may want to list all employees with their last name, employee id, and department name. However, the department name may be stored in the department table instead of the employee table. To perform queries of this type, we need to create *table joins*. The Visual Query Builder allows you to create joins using a simple drag-and-drop interface.

You begin a join operation by dragging the column name you want to join from the first table frame and dropping it on the target column name on the second table frame. When you drag the column name out of the first table frame and into the query workspace, the mouse pointer changes to , signifying that the column cannot be dropped inside the workspace. When you reach the target table frame, the mouse pointer changes to

, signifying that it can be dropped in that position. When you complete the join, a line is drawn in the query workspace linking the columns in the two table frames. If you move the table frames in the query workspace, the line is automatically redrawn to indicate the join condition.

The join criteria can be reviewed and edited by double-clicking on the line indicating the join. By default, the join is an inner join. If you want to change the join type to an outer join, for example, double-click on the join line. The **Join** dialog is displayed.

The **Table1** and **Column1** values identify the first table / column in the join. The **Table2** and **Column2** values identify the second table / column in the join. In case of an inner join, the order of table columns is not important, but it is important in case of outer joins.

Join Operator shows =. This specifies that the join is based on the equality of column values. You can specify a different join operator by selecting the corresponding option. For more information on various join operators, refer to the documentation of your database management system.

Join Type defaults to **Inner Join**. You can change it to other types of join supported by the database manager.

Some ODBC drivers support only one type of join known as inner join. Others support outer joins, but only one type of outer join known as left outer join. The **Join Type** options in the **Join** dialog displayed by Visual Query Builder reflect the capability of the driver to support different types of join. In case of dBASE, for example, only the **Inner**, **Left Outer** and **Right Outer** options are enabled.

A table join can specify more than two tables, or a join between two columns in the same table. A join in which columns from the same table are referred to is known as a *self-join*. In case of a self-join, the Visual Query Builder adds a prefix such as __1, __2, etc., to identify multiple instances of the same table. For example, if you create a listing of employees and their managers using the EmplId and MgrId columns in the Employee table, the columns will be identified as Employee.EmplId and Employee__1.MgrId.

Instead of specifying an inner join (which is the default), you can perform an outer join between the tables in a query. To create the outer join, you need to edit the join definition in the **Join** dialog. When you create a join by connecting two table frames, an inner join is created as default. To change the join type to outer join, double-click the line joining the table frames in the query workspace (or select the line by mouse-click and press ENTER). The **Join** dialog will be displayed.

To define the outer join, select the appropriate option button for **Join Type**. Only the outer join options supported by the ODBC driver are enabled and available for selection. After selecting the join type, choose **OK** to define the join. If you do not want to make a change, choose **Cancel**.

Since some ODBC drivers restrict the number of outer joins in a SELECT statement to one, you can define only one outer join condition in a query generated by the Visual Query Builder. If one outer join is defined, for all other joins in the query, the outer join option buttons will be disabled.



When you return to Visual Query Builder after defining an outer join, the line that connects the table frames is shown in red color.

Join - Example

Query: List the subject, title, ISBN, author and publisher name of books on SQL (In all examples included in the help file, we assume that the tables and columns exist as described).

In the sample database, the details of each book are maintained in the **TITLES** table. The information on the subjects covered by each book is maintained in the **SUBJECTS** table. The **ISBN** column is used as a link field between the two tables. To get the information required by this query, we would, therefore, need to specify that the **SUBJECTS** table be joined with the **TITLES** table using the **ISBN** column.

To build and test this query, perform the following steps:

1. Choose the **New** icon on the Visual Query Builder Toolbar to create a new query.
The **Add Table** dialog box will be displayed.
2. Double-click the **SUBJECTS** table to add it to the query. Alternatively, select the table name from the list of tables and choose the **Add** command button.
The fields in the table are displayed in the **SUBJECTS** table frame in the workspace.
3. Add the **TITLES** table to the query as described in Step 2.
The fields in the table are displayed in the **TITLES** table frame in the workspace.
4. Close the **Add Table** dialog using the **Close** command button.
5. Arrange the table frames in the workspace so that you can easily work with them.
6. Select the **ISBN** column in the **SUBJECTS** table by clicking on it. Then drag the column name across the workspace and drop it on the **ISBN** column in the **TITLES** table frame.
When you drag the column name out of the **SUBJECTS** table frame and into the query workspace, the mouse pointer changes to , signifying that the column cannot be dropped inside the workspace. When you reach the **TITLES** table frame, the mouse pointer changes to , signifying that it can be dropped in that position.
A line is drawn in the query workspace linking the **ISBN** columns in the two table frames. If you move the table frames in the query workspace, the line is automatically redrawn to indicate the join condition.
7. To view the join criteria, double-click the line joining the two columns.
The **Join** dialog is displayed.
The **Table1** and **Column1** values identify the first table / column in the join. The **Table2** and **Column2** values identify the second table / column in the join. In case of an inner join, the order of table columns is not important, but it is important in case of outer joins.
Join Operator shows =. This specifies that the join is based on the equality of column values. You can specify a different join operator by selecting the corresponding option. For more information on various join operators, refer to the documentation of your database management system.
Join Type defaults to **Inner Join**. You can change it to other types of join supported by the database manager.
8. Choose **OK** in the **Join** dialog.
9. Choose the **Subject** column from the **SUBJECTS** table to query.
10. Choose the **Title**, **ISBN**, **Author** and **Publisher** columns from the **TITLES** table to query.
11. In the **Criteria** row in the **Subject** column, type **'SQL'**. Include the single quotes.
12. Run the query.
13. The SQL statement for this query reads as follows:

```
SELECT Subject , Title , TITLES.ISBN , Author , Publisher
FROM   SUBJECTS , TITLES
WHERE ( SUBJECTS.ISBN = TITLES.ISBN )
      AND
      (( Subject = 'SQL' ) )
```

Note that the **ISBN** column in the SELECT statement's column list is qualified using the table name **TITLES**. Also note the use of the column qualifier for the **ISBN** column in the WHERE clause. This is required because the same column name is present in the **SUBJECTS** and **TITLES** tables. The Visual Query Builder automatically uses column qualifiers wherever required to identify column names unambiguously.

14. Save the query as **titlesub**.

Options Dialog


The **Options** dialog is used to:

- Remove duplicate records from the result set. If you want to do this, check the **Remove Duplicate Records** check box. This is the equivalent of adding the DISTINCT keyword to a SQL SELECT statement.
- Delimit column and table names. This may be needed to work with databases that allow embedded spaces in column and table names, or to use SQL reserved words in column and table names. To enable the option, check the **Always Quote Column and Table Names** check box.
- Enable or disable the validation of table joins. To enable the option, check the **Validate Joins** check box. If this option is enabled, the Visual Query Builder verifies if the columns to join have compatible data types. If the columns have incompatible data types, you will receive an error message.
- Enable or disable the validation of selection criteria. To enable the option, check the **Validate Criteria** check box. If this option is enabled, the Visual Query Builder verifies that the selection criteria specified are syntactically correct for a SQL SELECT statement.

Expression Dialog

Example

You can define expressions as part of a query. These can be arithmetic expressions that perform calculations on numeric data values, or string expressions that concatenate strings or create substrings. Note that string expressions are supported differently by different database management systems.

To define an expression, choose the  icon from the Toolbar. The **Expression** dialog box is displayed.

You should select at least one table before building an expression.

Visual Query Builder will assign a default name to each expression you define. You can change it by typing a different name in the **Expression Name** edit box.

The query can contain more than one expression. You select an expression to edit from the pulldown list.

The list of tables selected in the query will be shown in the **Tables** pulldown list. The columns of the table name shown in the **Tables** edit box are displayed in the **Columns** list. You can include any of the displayed columns as operands for the expression. To include columns from a different table, select the table from the **Tables** pulldown list and choose the required column from the **Columns** list.

In addition to column names, you can also include literal and numeric constants in the expression.

You can include the Addition (+), Subtraction (-), Multiplication (*) and Division (/) operators in an expression. You can change the precedence of these operators by including the operands in parentheses. To use any of the operators or parentheses in the expression, just double-click the item in the **Operators** list. It will be placed in the current cursor position in the **Expression** edit box.

The **Expression** edit box contains the definition of the expression. As you add more operands and operators, the **Expression** edit box will be automatically updated. You can directly edit the expression if you want to include literal or numeric constants.

You can also include the SQL aggregate functions in the expression: AVG, COUNT, MIN, MAX and SUM. Just select the function name from the **Functions** list. An **X** is placed as a place holder argument for the function. You have to replace it with the column name(s) on which the aggregate function is to be calculated.

In addition to SQL aggregate expressions, a number of built-in functions can also be included in the expression. Just pick the function name from the list displayed.

To save expression definitions and exit, choose **Save**. To exit without saving, choose **Done**. If you made changes, you are prompted to confirm an exit without save.

We create a simple expression in the example demonstrated for this topic. A more complex expression is demonstrated under the topic Specifying Grouping Criteria.

Expression - Example

Query: List the item number, ISBN, quantity, selling price and extended price of the books in order number 10011 (In all examples included in the help file, we assume that the tables and columns exist as described).

In the sample database, the information on books ordered is maintained in the **DETAILS** table. For this query, this table alone is needed.


To build and test the query, perform the following steps:

1. Choose the **New** icon from the Toolbar to create a new query.
The **Add Table** dialog will be displayed.
2. Add the **DETAILS** table to the query workspace. Close the **Add Table** dialog.
3. Add the **OrderId, ItemNo, ISBN, Quantity** and **SalePrice** columns to the query grid.
4. Choose the **Expr** command.
The **Expression** dialog box will be displayed.
5. Change the **Expression Name** to **Extended_Price**.
Extended_Price will be used as the name of the expression column in the query result.
6. Double-click the **Quantity** column in the **Columns** list.
The column name is automatically added to the **Expression** edit box.
7. Double-click the multiplication operator (*) in the **Operators** list.
The operator is appended to the **Quantity** column in the **Expression** edit box.
8. Double-click the **SalePrice** column in the **Columns** list.
The column name is appended to the multiplication operator in the **Expression** edit box.
9. Run the query.
You will see the list of books in order number 10011, along with the extended price of each order item.
10. The SQL statement for this query will read as follows:


```
SELECT ItemNo , ISBN , Quantity , SalePrice ,  
      ( DETAILS.Quantity * DETAILS.SalePrice ) as Extended_Price  
FROM   DETAILS  
WHERE  ( OrderId = 10011 )
```
11. Save the query as **extprice**.

The queries you create using expressions may not be portable across all databases.

SQL Window

The SQL Window displays the SQL SELECT statement associated with the current query. To bring up the SQL Window, choose the  icon from the Toolbar. As you add or change query columns, selection criteria, grouping, or sorting criteria, the SQL Window is automatically updated. Viewing the SQL statement gives you immediate feedback about the query design and also helps learning the SQL syntax.

Result Window

You can run the query generated by the Visual Query Builder by choosing the  icon from the Toolbar. The query results are displayed in the Result Window. This helps you verify that query columns, selection criteria, grouping, and sorting criteria have been correctly specified for the query.

