APPENDIX

Using SQL Queries in Crystal Reports

In this appendix

The SQL Commands feature was introduced in Chapter 1, "Creating and Designing Basic Reports." For those users who are unfamiliar with *SQL (Structured Query Language)*, this appendix serves as an introduction and helps enable the creation of SQL commands. For those users who are familiar with SQL, this appendix serves as a refresher with some important tips pertaining to the use of SQL with the new SQL Commands feature. This chapter provides the following:

- A review of SQL Commands
- An introduction to SQL

REVIEW OF SQL COMMANDS

With reports based on tables, views, or stored procedures, Crystal Reports does the background work of generating a database query. This query incorporates the fields you have used in the report, any sorting or filtering you've applied, and even some calculations. This is one of the strengths of Crystal Reports—you don't need to be an expert at writing SQL to use the product. All that complexity is abstracted away from the user designing the report. However, sometimes the person developing the report is familiar with the SQL language, and perhaps is also the database administrator. In situations like this, people often want to write their own query for several reasons, including the following:

- An already defined query, which has the required fields, is in use elsewhere.
- The user wants to optimize her query beyond what Crystal Reports provides out-of-the-box.
- The user wants to perform a complex query that is beyond what Crystal Reports automatically generates; for example, a union query.

The SQL Commands feature is meant to address these needs. Rather than adding a table or view to a report, you can add a SQL command. This command represents a SQL query that you will type in. After this SQL command is created, it is treated just like a table in that it contains fields that can be used in the report and can be linked to other tables or SQL commands.

AN INTRODUCTION TO SQL

As its name implies, SQL is used to express a database query. SQL has facilities for defining which fields should be returned from the query, if and how the query should be filtered and sorted, and so on. Although SQL is an industry standard language, various specific versions and editions of the standard are implemented by SQL-based databases. Crystal Reports does not use just a single syntax, but rather is robust enough to handle most major SQL language derivations. The rest of this appendix walks you through the SQL Language and points out specific areas that are of concern to Crystal Reports. Although it doesn't focus on a specific version of SQL, it does point out differences where appropriate.

The SELECT Statement

Even though the name implies that SQL is only about querying databases, most implementations also enable you to insert, delete, and update records inside the database. Each of these distinct actions has its own command: `SELECT` (query), `INSERT`, `UPDATE`, and `DELETE`. Although SQL commands allow any valid SQL statement that returns records to be used, `SELECT` statements are generally the only statements to be used. However, there are situations in which other statements can be used in addition to a `SELECT` statement. One example of this is running an `INSERT` statement to create a record to log the fact that the report is being run. This section focuses on describing the `SELECT` statement from SQL.

A basic `SELECT` statement has the following syntax:

```
SELECT field-list
FROM table-list
```

`SELECT` statements always begin with the word `SELECT`. The general convention is to capitalize all SQL keywords used in the query to make it clear which is SQL and which is a table or field name. The list of fields to include is a comma-separated list of field names, such as "Name, Age, Gender." To include all fields in the specified table(s), use an `*` instead of listing individual field names. If the name of a field contains a space, the field name should be surrounded by a quote character (`'field name'`). Various SQL implementations allow different quotes, but most of them support `'` (single quote) as a quote character. The list of tables follows the same convention: They are separated by commas and are optionally enclosed in a quote. Any extra whitespace or carriage returns are usually ignored by the database. The following is a sample SQL statement using the Xtreme Sample Database:

```
SELECT 'Customer Name', City, Country
FROM Customer
```

Notice that quotes were only used for the Customer Name field because it was the only field with a space in the name. However, as a general convention, quote all your field and table names to be safe. The same statement could be written like this:

```
SELECT 'Customer Name', 'City', 'Country'
FROM 'Customer'
```

Depending on the type of database, table names can also be prefixed with the associated database name, for example, MyDatabase.MyTable. When using a qualified name such as this, you need to quote both names separately; that is, `'MyDatabase'.'MyTable'`.

NOTE

When you're using a SQL command in Crystal Reports, the fields that you specify in the field-list part of the `SELECT` statement determine which fields will be available to you inside your report. Although it's easy to use a `SELECT * ...` statement, keep in mind that you could be bringing back fields that aren't used and increasing processing time and required bandwidth. It's better to specify individual fields. You can always add or remove a field after the SQL command is created by opening the Database Expert, right-clicking on the Command object, and selecting Edit Command from the context menu.

In the previous examples, data was being returned for each customer. However, if you wanted to return a list of countries, you might use a query such as the following:

```
SELECT 'Country'
FROM 'Customer'
```

Although this wouldn't return incorrect results, it would return redundant results because there is more than one record that contains the same country name. To work around this, use the `DISTINCT` keyword, which filters out all duplicate records:

```
SELECT DISTINCT 'Country'
FROM 'Customer'
```


FILTERING RECORDS

By learning a basic `SELECT` statement, you have the capability to return any or all fields. But so far, the query would return all records stored in that table. This section builds on what you've learned up to now by introducing a new clause in the SQL statement. If you're not sure why you would want to filter records, consider that an "average" corporate data source might contain millions of records of data, and without being more specific in a query, you are putting an undue load on the database server as well as overwhelming business users with more data than they need.

The `WHERE` clause enables you to specify which records should be included in the query. If the `WHERE` clause is omitted (as it has been in the examples thus far), all records from the table are returned. Specifying a `WHERE` clause can limit these records to a more relevant subset. The syntax of a SQL statement with a `WHERE` clause is as follows:

```
SELECT field-list
FROM table-list
[WHERE condition]
```

NOTE

Square brackets that enclose any component of the SQL statement indicate that component is optional and need not be included in the SQL statement.

The condition can be any equality expression. Fields from the table can be used in the condition, as well as text literals and numbers. Let's look at a few examples:

```
SELECT 'Customer Name', 'City'
FROM 'Customer'
WHERE 'Country' = 'USA'
```

The preceding SQL statement returns all customers who have a `Country` of `USA`. Notice that in this statement a text literal is used (`'USA'`). Text literals are surrounded by a text delimiter. The most common delimiter is the single quote, as used here.

Conditions can be combined together with `AND`s and `OR`s, as shown in the following example:

```
SELECT 'Order ID', 'Order Date'
FROM 'Orders'
```

```
WHERE 'Order Amount' > 2000 AND
            'Customer ID' = 123
```

NOTE

Sometimes it's appropriate to use a SQL statement that has a hard-coded (static) number or string. However, it's often more common to use parameters in the place of such values. That way, the report can be reprocessed with different values showing diverse information without having to modify the SQL Command each time a change is needed. To create a parameter in SQL syntax, click the Create button in the Create SQL Command dialog and substitute the parameter name in place of the hard-coded value.

SORTING RECORDS

Like filtering, sorting can be performed by Crystal Reports on your local workstation. However, it's always faster to have the database itself perform the operation because a typical database server has far more processing power than your desktop PC. This section introduces another clause to the SQL statement that enables you to specify the order in which the records are returned.

The `ORDER BY` clause is used to specify sorting. The syntax is as follows:

```
SELECT field-list
FROM table-list
[WHERE condition]
[ORDER BY field-list [ ASC ¦ DESC ] ]
```

The `ORDER BY` clause comes last in the SQL statement and is followed by a comma-separated list of fields. The records returned from the query will be sorted first by the first field specified, and then by the second, and so on. By default, fields are sorted in ascending order (from smallest to largest, or A to Z); but by adding `ASC` or `DESC` after the field name, you can specify either ascending or descending (largest to smallest, or Z to A) sort order.

The following SQL statement sorts the records by country, and then by region:

```
SELECT *
FROM 'Customer'
ORDER BY 'Country' ASC, 'Region' ASC
```

The preceding example is sorting alphabetically. The following example shows where sorting is done on a numeric field. This query returns a list of customers in the order of highest sales first.

```
SELECT 'Customer Name', 'Last Year's Sales'
FROM 'Customer'
ORDER BY 'Last Year's Sales' DESC
```

JOINING MULTIPLE TABLES

So far, we've only used a single table, but of course multiple tables can be used. You might have already tried a statement like this:

```
SELECT 'Customer Name', 'Order ID'
FROM 'Customer', 'Orders'
```

This might seem correct initially, but this query most likely won't return what you are looking for. Although only 2,192 records are in the Orders table, this query will return more than 500,000 records and, with a larger database, could actually bring down the database server! This is because for each record in the Customer table, the entire set of records in the Orders table is included. In other words, the database doesn't know how to match up the records between the tables. If more than one table is used, a join should be applied that indicates how to match up the tables. There are various syntaxes for joins, but the simplest is to add a `WHERE` clause to the SQL statement (shown as follows), which produces an equal join:


```
SELECT field-list
FROM table1, table2
WHERE table1.field = table2.field
```

This type of join applied to the previous sample query would look like this:

```
SELECT 'Customer Name', 'Order ID'
FROM 'Customer', 'Orders'
WHERE 'Customer'.'Customer ID' = 'Orders'.'Customer ID'
```

Notice that because there is a Customer ID field in both the Customer and Orders tables, when that field is referenced in the `WHERE` clause, it is prefixed with the table name so as not to be ambiguous.

ALIASING

One beneficial feature of SQL is the capability to give fields and tables more meaningful names. Often fields are defined in the database with non-meaningful names such as ACTID instead of Account ID, and it would be useful to rename, or alias, this name.

Aliasing is straightforward: After the field that you want to alias, simply append `'AS` *`field-name`*`'`, where *field-name* is the new name for the field. Here's a working example:

```
SELECT 'Customer Name', 'Region' AS 'State'
FROM 'Customer'
WHERE 'Country' = 'USA'
```

In this example, because the records are being filtered to only include customers from the `USA`, it can be inferred that that the `Region` field will contain the `State` (where other countries such as Canada might use the `Region` field for the province). Because of this, the field is aliased to `State`. Note that the alias name need not be contained in quotes unless it has a space, but as stated earlier, it's good practice to always quote field names.

CALCULATED FIELDS

It's often a requirement to display data on the report that doesn't exist directly in the database—that is, data inferred or calculated based on other fields in the database. Although Crystal Reports provides a full formula language for defining these "formulas," when using SQL only, you need to follow its rules and limitations. SQL does have the capability to handle basic expressions like this. An expression, or calculated field, is specified in the `SELECT` part of the SQL statement just like any other field. Consider the following example, which uses an expression to concatenate a first and last name field together:

```
SELECT 'Customer Name', 'Contact First Name' + ' ' + 'Contact Last Name'
FROM 'Customer'
```

If you were to use this SQL statement in a SQL command, the correct field values are returned; however, the calculated field would be named something slightly cryptic like Expr1001. It's clear to you that this field represents a Contact Name, but the database can't easily infer that. To correct this problem, draw on the aliasing concept explained in the previous section. The corrected SQL statement is here:


```
SELECT 'Customer Name',
       'Contact First Name' + ' ' + 'Contact Last Name' AS 'Contact Name'
FROM 'Customer'
```

In addition to textual expressions, you can perform mathematical expressions as well. The following SQL statement uses a calculated field to determine the tax paid based on sales:

```
SELECT 'Customer Name', 'Last Year's Sales',
       'Last Year's Sales' * 0.07 AS 'Tax Paid'
FROM 'Customer'
```

For more information on what kind of expressions can be used in your SQL command, consult the documentation for your database.

UNION QUERIES

In the Xtreme Sample Database, each table represents a certain type of object, but often multiple tables represent the same type of object. For example, rather than having a single table called Orders, you might have multiple tables called Orders2001, Orders2002, Orders2003, which each contain the orders for a particular year as indicated by the table name. If you only want to report off one of those tables at a time, you don't need to do anything special. But, if you'd like to consolidate those together into a single query result, you must use a union query.

TIP

Union queries were not inherently supported by Crystal Reports in previous versions. However, the introduction of SQL Commands in Crystal Reports version 9 enables you to use this feature fully.

The syntax for a union query is as follows:

```
SELECT statement
[ UNION
SELECT statement ]
```

Here is a SQL statement with a `UNION` clause combining some fictitious order tables:

```
SELECT * FROM 'Orders2001'
UNION
SELECT * FROM 'Orders2002'
UNION
SELECT * FROM 'Orders2003'
```

These tables can be unioned together because they have the same table structure. You are not able to perform a union on two tables with different fields.

GROUPING

Grouping enables records to be grouped together based on a specified field, and then summarized using a given summary operation. Note that grouping in a SQL command will not allow a drill-down to the detail records. The syntax for grouping is as follows:


```
SELECT field-list
FROM table-list
[WHERE condition]
[GROUP BY field-list]
[ORDER BY field-list [ ASC ¦ DESC ] ]
```

The following example groups all customers by country and summarizes the sales:

```
SELECT 'Country', SUM('Last Year's Sales') AS 'Total Sales'
FROM 'Customer'
GROUP BY 'Country'
```

Two components to grouping exist in a SQL statement. The first is the summary operation—that is, `SUM`, `COUNT`, `AVG`, and so on. This operation determines which field will be summarized and in what way. The second component is the `GROUP BY` clause, which specifies for which field the data should be summarized—in other words, on which field the data should be grouped.