



CHAPTER

3

# Oracle SQL

## IN THIS CHAPTER:

Basic SELECT Statement

SELECT List

FROM Clause

Join Types

Filtering

Ordering

Grouping

Group Filtering

Operators

Oracle Comparison Conditions

Aggregation

Working with Dates

Working with Strings

Working with Numbers

Other Common Functions

Pseudo Columns

Subqueries

The purpose of this chapter is to cover Oracle SQL basics as they relate to Crystal Reports. A generic knowledge of the SQL SELECT statement is assumed and only Oracle-specific, new to 9i, or otherwise uncommon options are discussed in any detail. New join options available in Oracle 9i are also covered. The SELECT list, FROM clause, WHERE clause, ORDER BY clause, and GROUP BY clause are explained, and how they are generated from Crystal Reports is shown. Oracle operators, comparison conditions, and aggregation functions are covered, and working with Oracle dates, strings, and numbers is described. Commonly used Oracle-specific functions and pseudo columns are covered, and the use of subqueries will also be discussed.

---

## Basic SELECT Statement

The basic SELECT statement can take one of the following two forms. Refer to Oracle documentation for a full list of possible clauses and options. Various clauses of the SELECT statement are discussed in more detail in following sections.

The first form uses the WHERE clause to create the table joins:

```
SELECT [list of columns or expressions separated by commas]
      FROM [list of tables separated by commas]
      WHERE [join conditions and filtering conditions]
      GROUP BY [optional list of columns to group by]
      ORDER BY [optional list of columns to order by]
      HAVING [optional group filtering conditions]
```

The second form uses the new Oracle 9i join syntax:



### NOTE

---

*The second form is not available in Oracle 8i or previous versions.*

```
SELECT [list of columns or expressions separated by commas]
      FROM [tables with join expressions]
      WHERE [filtering conditions]
      GROUP BY [optional list of columns to group by]
      ORDER BY [optional list of columns to order by]
      HAVING [optional group filtering conditions]
```

Aliases can be created for columns, expressions, or tables simply by inserting the desired alias immediately following the column name, expression, or table name. The keyword AS may be used for clarity preceding the alias. Column or expression aliases are returned as the column name and table aliases are used as shortcuts for the full schema qualified table name, within the query for linking, or when it is otherwise necessary to distinguish which table a field belongs to.

It is rare for Oracle table, view, or column names to be defined with mixed-case characters. They are usually defined in uppercase only. If the objects are defined in Oracle using uppercase, they can be listed in the query in upper, lower, or mixed case and the query will succeed. If an object name is defined with mixed case in Oracle, then the object name must be enclosed in double quotes in the SELECT statement; otherwise, Oracle will return an error saying that the object cannot be found.

The keyword DISTINCT can be added before the SELECT list to return only one row if there are multiple rows with the same column values. The DISTINCT keyword applies to the entire SELECT list and can cause increases in processing time due to the comparing and filtering that must be done. Avoid using DISTINCT if it is not required.

Except for omission of the HAVING clause, Crystal Reports constructs SELECT statements of the first form when you use the Crystal experts to create a report. Crystal always creates aliases for tables where the alias is the table name without the schema name and will always prefix column names with the alias it defined for the table. Crystal also always encloses object names with double quotes. If you want to return distinct records, choose the Database menu item and then the Select Distinct Records option, and DISTINCT will be added to the query.

---

## SELECT List

The SELECT list is simply a list of columns or expressions that the user wishes to return. Columns are specified by using the column name as defined in the table. A table and/or schema qualifier or alias should be added to the column name with dot notation if it is needed to distinguish the correct column from other like-named columns from other tables in the query. Expressions are valid computations resulting in a single value. Expressions can be single row or summary level if the corresponding GROUP BY clause is included and can include functions, but not procedures. An expression can even be an entire SELECT statement as long as it returns only one value.

The asterisk can be used as a shorthand symbol that means all columns. For example, if you have a command like the following, all columns from the Employee table will be selected.

```
SELECT * FROM Employee
```

The asterisk can be prefixed with a table name or table alias to indicate all columns from a particular table.

```
SELECT Product.*, d.Order_ID
FROM Product JOIN Orders_Detail d
ON (Product.Product_Id=d.Product_Id)
```

---

## FROM Clause

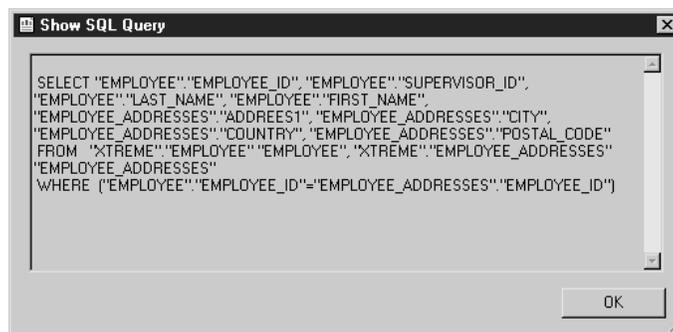
The tables listed in the FROM clause can be database tables or views or entire SELECT statements. If a SELECT statement is used it is called an in-line view.

---

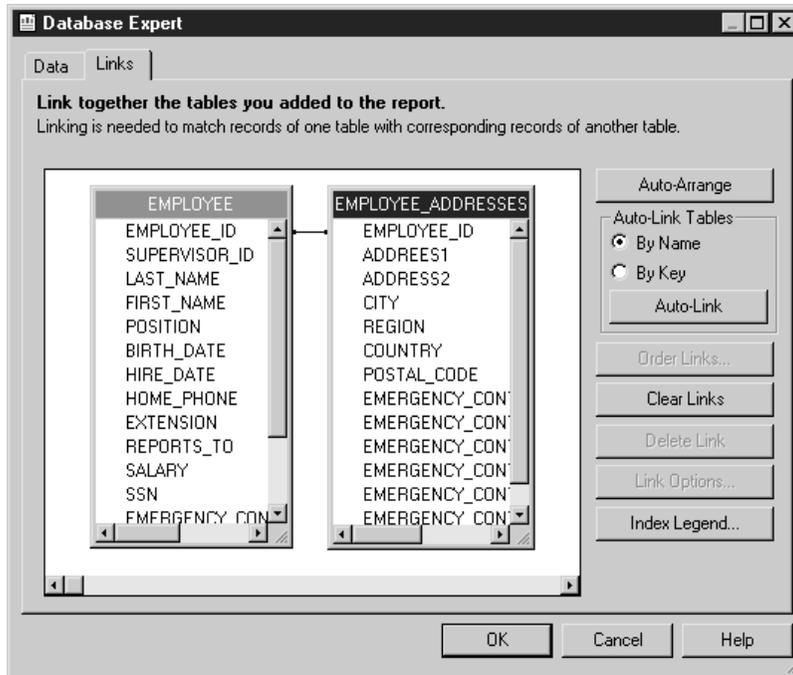
## Join Types

One of the most basic and valuable features of a relational database is the ability to join two tables together. Crystal Reports uses the Links tab of the Database Expert to enable the user to establish links via a GUI interface. In a simple linking example such as that shown in Figure 3-1, Crystal will generate an equal inner join.

The join that Crystal generates can be seen by selecting the Database menu item, then the Show SQL Query option. The SQL is shown here:



It is the ( "EMPLOYEE" . "EMPLOYEE\_ID" = "EMPLOYEE\_ADDRESSES" . "EMPLOYEE\_ID" ) part that implements the join.



**Figure 3-1** Database Expert simple join

Joins have several possible types. A join might be an inner join, a left outer join, a right outer join, or a full outer join. The link type might be equal, greater than, greater than or equal, less than, less than or equal, or not equal. The most common joins are equal inner joins and equal left outer joins. In an equal inner join, records from each table are returned if the values in the join columns are equal. Any records from either table that do not have exact matches in the other table are ignored. In an equal left outer join, all records from the left-hand table are returned and any records from the right-hand table whose join columns match are merged into the matching row. For left-hand rows with no match, null column values are appended to the row. Equal right outer joins are identical to equal left outer joins except that all rows from the right-hand table are returned and matching left-hand table rows are merged. In equal full outer joins, rows with identical values in the join columns from each table are merged, rows from either table that did not have matches are also returned, and the missing columns are populated with null values. It is common to assume an equal join and omit the word equal when describing joins.

Nonequal joins are similar to equal joins except that more than one row from the left-hand table may be joined to more than one row from the right-hand table. For example,

say that you want to know, for Product\_ID 4103 for the month of May for each ship\_date the quantity to ship on that date and the total quantity shipped up to and including that date. You need to display the sum of the quantity shipped on that date, which is straightforward, but you also need to sum the quantities shipped before that date. Breaking down the process, first you need the quantities by date for the product for May:

```
SQL> SELECT ship_date, SUM(quantity) quantity
  2   FROM Orders JOIN Orders_Detail USING (Order_Id)
  3   WHERE product_id=4103
  4     AND ship_date BETWEEN TO_DATE('01-MAY-02', 'DD-MON-YY')
  5                           AND TO_DATE('31-MAY-02', 'DD-MON-YY')
  6   GROUP BY ship_date;
```

SHIP_DATE	QUANTITY
02-MAY-02	2
04-MAY-02	5
09-MAY-02	3

Then you need to join each record to every record that shipped earlier or on the same date:

```
SQL> SELECT a.ship_date, a.quantity, b.ship_date, b.quantity
  2   FROM (SELECT ship_date, SUM(quantity) quantity
  3         FROM Orders JOIN Orders_Detail USING (Order_Id)
  4         WHERE product_id=4103
  5           AND ship_date BETWEEN TO_DATE('01-MAY-02', 'DD-MON-YY')
  6                               AND TO_DATE('31-MAY-02', 'DD-MON-YY')
  7         GROUP BY ship_date) a,
  8   (SELECT ship_date, SUM(quantity) quantity
  9         FROM Orders JOIN Orders_Detail USING (Order_Id)
 10        WHERE product_id=4103
 11          AND ship_date BETWEEN TO_DATE('01-MAY-02', 'DD-MON-YY')
 12                              AND TO_DATE('31-MAY-02', 'DD-MON-YY')
 13        GROUP BY ship_date) b
 14  WHERE a.ship_date >= b.ship_date;
```

SHIP_DATE	QUANTITY	SHIP_DATE	QUANTITY
09-MAY-02	3	09-MAY-02	3
09-MAY-02	3	04-MAY-02	5
09-MAY-02	3	02-MAY-02	2
04-MAY-02	5	04-MAY-02	5

04-MAY-02	5	02-MAY-02	2
02-MAY-02	2	02-MAY-02	2

This result shows the May 9 record joined to the May 2, May 4, and May 9 records, the May 4 record joined to the May 2 and May 4 records, and the May 2 record joined to itself.

Finally, you need to sum the “b” values to get the month to date quantities:

```
SQL> SELECT a.ship_date, MIN(a.quantity) "Shipped on Ship Date",
2      SUM(b.quantity) "Shipped up to Ship Date"
3 FROM (SELECT ship_date, SUM(quantity) quantity
4      FROM Orders JOIN Orders_Detail USING (Order_Id)
5      WHERE product_id=4103
6      AND ship_date BETWEEN TO_DATE('01-MAY-02','DD-MON-YY')
7      AND TO_DATE('31-MAY-02','DD-MON-YY')
8      GROUP BY ship_date) a,
9      (SELECT ship_date, SUM(quantity) quantity
10     FROM Orders JOIN Orders_Detail USING (Order_Id)
11     WHERE product_id=4103
12     AND ship_date BETWEEN TO_DATE('01-MAY-02','DD-MON-YY')
13     AND TO_DATE('31-MAY-02','DD-MON-YY')
14     GROUP BY ship_date) b
15 WHERE a.ship_date>=b.ship_date
16 GROUP BY a.ship_date
17 ORDER BY a.ship_date;
```

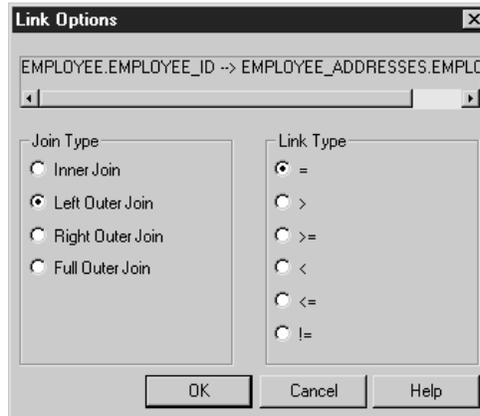
SHIP_DATE	Shipped on Ship Date	Shipped up to Ship Date
02-MAY-02	2	2
04-MAY-02	5	7
09-MAY-02	3	10

Note that this result can be obtained in a simpler manner by using the Oracle SQL for Analysis functions. See the “Analysis” section in Chapter 4 for more details.

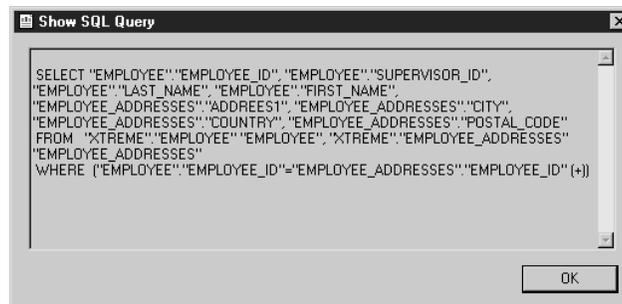
## Pre-Oracle 9i Joining

Prior to Oracle version 9i, all linking was done in the WHERE clause. A clause would be written in the form `Table1.LinkField [operator] Table2.LinkField`, where the operator could be `=`, `<=`, `<`, `>`, `>=`, `<>`, `!=`, and so on. To accomplish a left outer or right outer join, the (+) symbol was added to the appropriate side of the condition. A full outer join could not be done without using a UNION operation.

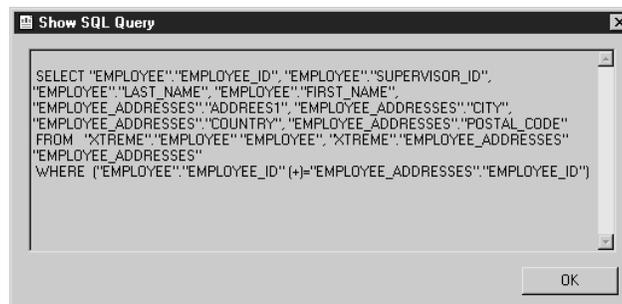
In Crystal Reports, to change the link type, go to the Database Expert, Links tab, select the link and click Link Options, or right-click the link and select Link Options. The Link Options dialog will be displayed, as shown here:



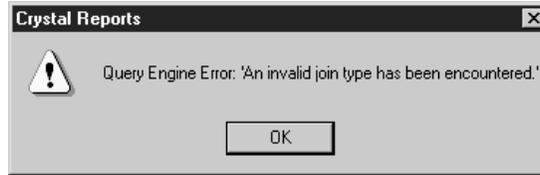
The next illustration shows the SQL for a left outer join using pre-9i syntax as it is generated in Crystal Reports.



The next illustration shows a right outer join using pre-9i syntax as it is generated in Crystal Reports.

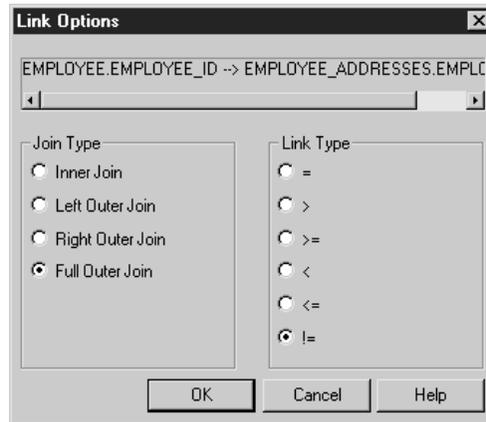


Choosing Full Outer Join with an Oracle 9i database results in the error shown here:



The Full Outer Join radio button is available when using the native driver even though choosing it causes an error. It is not available with the ODBC drivers or the OLE DB drivers.

Link types can be changed from the Link Options dialog as well. The less than, greater than, not equal, and so on, types work as expected. Join types other than equal are rarely used, but they can be the proper answer for some problems, as shown in the preceding example. Note that nonequal joins are not the same as one-to-many joins, as a one-to-many join only joins if the condition field is equal in both tables. The following illustration shows a not equal join.



In this case, each Employee record would be joined to each Employee\_Addresses record where Employee.Employee\_ID is not equal to Employee\_Addresses.Employee\_ID. This is not a sensible join.

## 9i Joining

Oracle 9i implements SQL 1999 compliant join operations. You can use this new join terminology in Crystal Reports SQL Commands, Oracle views, and Oracle Stored Procedures, but the Crystal Report designer still uses Oracle 8 syntax.

The execution plan that is generated using the new join syntax may differ from the plan that would be created using the old syntax. In general, full outer joins appear to be optimized with the new syntax, right and left outer joins seem to be equivalent, and inner joins may sometimes be less efficient. If large tables are being joined, a comparison of both methods is recommended to determine the optimal execution plan. Since Oracle recommends using the new syntax, it is expected that any deficiencies in the plans generated will be corrected over time. For now, it may be possible to use optimizer hints to cause the plan created by the new syntax to match the plan created by the old syntax.

## Cross Join

A cross join produces a Cartesian product of the tables if no conditions are added to the WHERE clause:

```
SELECT "EMPLOYEE"."EMPLOYEE_ID", "EMPLOYEE"."FIRST_NAME",
       "EMPLOYEE"."LAST_NAME", "EMPLOYEE_ADDRESSES"."CITY",
       "EMPLOYEE_ADDRESSES"."COUNTRY"
FROM "XTREME"."EMPLOYEE" "EMPLOYEE"
     CROSS JOIN
     "XTREME"."EMPLOYEE_ADDRESSES" "EMPLOYEE_ADDRESSES"
```

This statement results in every row from Employees being joined to every row in Employee\_Addresses. Cross joins are rarely useful and are usually the result of forgetting to add the join criteria to the statement.

## Natural Join

A natural join automatically joins the two tables on all fields with the same name and type:

```
SELECT "EMPLOYEE_ID", "EMPLOYEE"."FIRST_NAME",
       "EMPLOYEE"."LAST_NAME", "EMPLOYEE_ADDRESSES"."CITY",
       "EMPLOYEE_ADDRESSES"."COUNTRY"
FROM "XTREME"."EMPLOYEE" "EMPLOYEE"
     NATURAL JOIN
     "XTREME"."EMPLOYEE_ADDRESSES" "EMPLOYEE_ADDRESSES"
```

This statement will join Employee to Employee\_Addresses on the Employee\_ID field from each table using an equal inner join, since that is the only column in both tables of the same name.

Note that for a natural join or a JOIN USING, you cannot use a schema name in the SELECT list for the join field. Here you can use just "EMPLOYEE\_ID" with no schema qualifier.

## JOIN USING

The JOIN USING syntax lets you pick which columns to use to implement the join. The columns must have the same name, as in a natural join, but you can list only those fields that you want to join on and exclude any other fields that might be named the same. A JOIN USING clause results in an equal inner join:

```
SELECT "EMPLOYEE_ID", "EMPLOYEE"."FIRST_NAME",
       "EMPLOYEE"."LAST_NAME", "EMPLOYEE_ADDRESSES"."CITY",
       "EMPLOYEE_ADDRESSES"."COUNTRY"
FROM "XTREME"."EMPLOYEE" "EMPLOYEE"
     JOIN
     "XTREME"."EMPLOYEE_ADDRESSES" "EMPLOYEE_ADDRESSES"
     USING ("EMPLOYEE_ID")
```

## JOIN ON

The JOIN ON syntax allows you to specify join fields from the two tables where the column names may not match:

```
SELECT "EMPLOYEE"."EMPLOYEE_ID", "EMPLOYEE"."FIRST_NAME",
       "EMPLOYEE"."LAST_NAME", "EMPLOYEE_ADDRESSES"."CITY",
       "EMPLOYEE_ADDRESSES"."COUNTRY"
FROM "XTREME"."EMPLOYEE" "EMPLOYEE"
     JOIN
     "XTREME"."EMPLOYEE_ADDRESSES" "EMPLOYEE_ADDRESSES"
     ON ("EMPLOYEE"."EMPLOYEE_ID" =
         "EMPLOYEE_ADDRESSES"."EMPLOYEE_ID")
```

In this case, you must qualify the joined field name with a table name or alias when using it in the SELECT list. You can include anything in the ON clause that you would have previously used in a WHERE clause. However, it is beneficial to keep filtering clauses in the WHERE clause and joining clauses in the ON clause for clarity.

## OUTER

OUTER JOINS can be specified in Oracle 9i using the OUTER keyword. You no longer need to use the (+) operator:

```

SELECT "EMPLOYEE"."EMPLOYEE_ID", "EMPLOYEE"."FIRST_NAME",
       "EMPLOYEE"."LAST_NAME", "EMPLOYEE_ADDRESSES"."CITY",
       "EMPLOYEE_ADDRESSES"."COUNTRY"
FROM "XTREME"."EMPLOYEE" "EMPLOYEE"
LEFT OUTER JOIN
"XTREME"."EMPLOYEE_ADDRESSES" "EMPLOYEE_ADDRESSES"
ON ("EMPLOYEE"."EMPLOYEE_ID"=
    "EMPLOYEE_ADDRESSES"."EMPLOYEE_ID")

```

The preceding statement results in a left outer join. The OUTER keyword can be omitted if desired.

```

SELECT "EMPLOYEE"."EMPLOYEE_ID", "EMPLOYEE"."FIRST_NAME",
       "EMPLOYEE"."LAST_NAME", "EMPLOYEE_ADDRESSES"."CITY",
       "EMPLOYEE_ADDRESSES"."COUNTRY"
FROM "XTREME"."EMPLOYEE" "EMPLOYEE"
RIGHT OUTER JOIN
"XTREME"."EMPLOYEE_ADDRESSES" "EMPLOYEE_ADDRESSES"
ON ("EMPLOYEE"."EMPLOYEE_ID"=
    "EMPLOYEE_ADDRESSES"."EMPLOYEE_ID")

```

The preceding statement results in a right outer join.

```

SELECT "EMPLOYEE"."EMPLOYEE_ID", "EMPLOYEE"."FIRST_NAME",
       "EMPLOYEE"."LAST_NAME", "EMPLOYEE_ADDRESSES"."CITY",
       "EMPLOYEE_ADDRESSES"."COUNTRY"
FROM "XTREME"."EMPLOYEE" "EMPLOYEE"
FULL OUTER JOIN
"XTREME"."EMPLOYEE_ADDRESSES" "EMPLOYEE_ADDRESSES"
ON ("EMPLOYEE"."EMPLOYEE_ID"=
    "EMPLOYEE_ADDRESSES"."EMPLOYEE_ID")

```

The preceding statement results in a full outer join. Full outer joins were not possible prior to Oracle 9i without using a UNION operation.

---

## Filtering

Restricting the records returned based on some selection criteria is done in the WHERE clause. The Crystal Reports Select Expert translates the user's choices into expressions in the WHERE clause. For complex filtering, the selection formula can be modified manually.

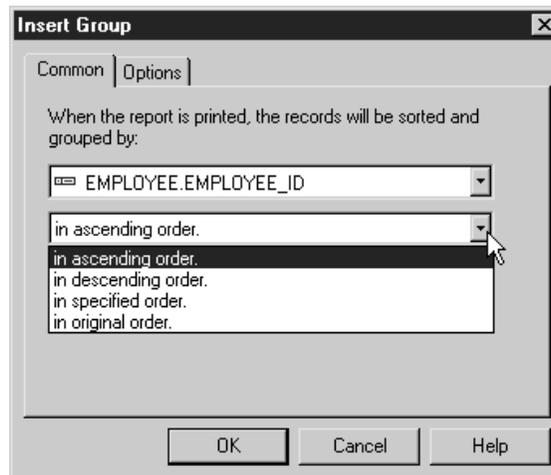
## Ordering

Ordering the records returned is implemented using the ORDER BY clause. Each field is sorted in the order specified (by default, the sort order is ascending, but a descending sort order can be specified).

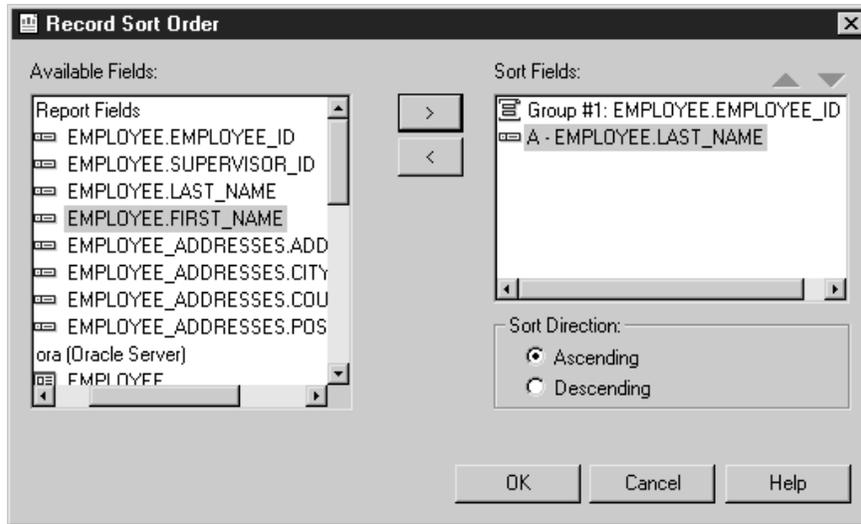
Crystal Reports will generate the ORDER BY clause depending on any existing group's Insert Group Options and the Record Sort Order options, as shown in Figure 3-2 and Figure 3-3. However, choosing the group option to sort in original order will have no impact because Oracle will sort the GROUP BY fields in ascending order by default. If the Sort In Specified Order is selected, then the sort cannot be done on the server and will be done locally.

## Grouping

Grouping is accomplished using the GROUP BY clause, which also performs a sort on the grouped fields. You must use a GROUP BY clause if you wish to use any aggregation functions such as SUM. Adding a group in Crystal Reports will not necessarily add a GROUP BY clause to the SQL query. An Oracle SELECT statement containing a GROUP BY clause will return data only at the grouped level; it will not return any detail rows. If a Crystal Report has a group and the detail rows are suppressed, Crystal will add a GROUP BY clause to the SQL query. If multiple



**Figure 3-2** *Insert Group Common Options*



**Figure 3-3** *Record Sort Order*

groups exist and detail rows are suppressed, Crystal will add a GROUP BY clause for the innermost not suppressed group.

---

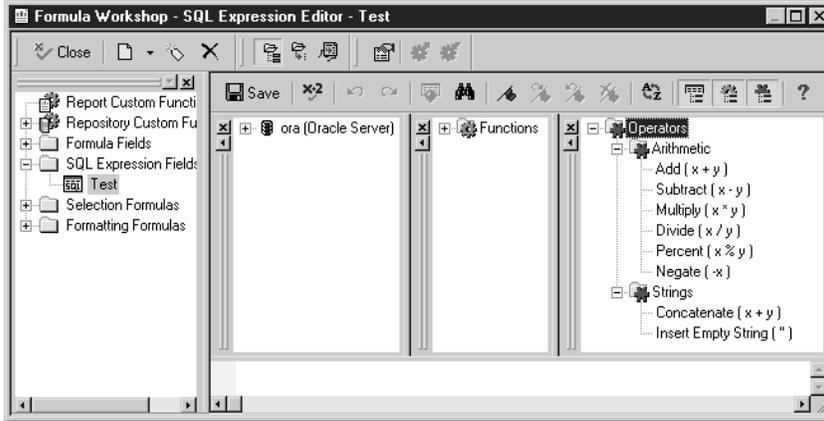
## Group Filtering

Oracle SQL supports using the HAVING clause to specify filtering at the group level. The HAVING clause is discussed in more detail in Chapter 4. Crystal Reports allows group selection formulas, but these are not translated into the SQL query and passed to the server. Group filtering that is done with Crystal options will be done locally.

---

## Operators

Oracle SQL operators are used to create complex expressions. The available arithmetic operators are + (addition), - (subtraction), \* (multiplication), and / (division). The concatenation operator is ||. Crystal Reports shows the Oracle operators in the SQL Expression Editor in the Operators box, as shown here:



There are some errors in the Crystal listing. The % operator is shown, but it is not a valid Oracle operator and any SQL Command containing it will fail. Also, the + is shown as the concatenation operator, but it will only work for numeric addition, not for string concatenation; you must use the double pipes (||) for concatenation. The \* is listed as the multiplication operator. This is not correct, but Crystal converts it to × in the SQL query, so it is not a problem.

---

## Oracle Comparison Conditions

Comparison conditions are used in the WHERE clause or HAVING clause to compare one expression to another. In addition to the usual =, <>, <, >, <=, >=, you can also use IS NULL, IS NOT NULL, LIKE, BETWEEN, NOT BETWEEN, IN, NOT IN, and EXISTS. Comparison conditions can be joined with AND or OR and negated with NOT.

Table 3-1 shows how the Crystal condition is translated into an Oracle WHERE clause when using the Crystal Reports Select Expert to create selection formulas. The strings used in the LIKE and NOT LIKE conditions can contain the Crystal wildcard characters ? and \*, which are translated to the corresponding Oracle wildcard characters \_ and %.

To use Oracle's IN, BETWEEN, or EXISTS comparison conditions in the WHERE clause, you must use a SQL Command, view, or stored procedure.

---

## Aggregation

If a GROUP BY clause is specified in the query, then aggregation functions can be used. Table 3-2 shows Crystal's summary functions and how they are translated into

Select Expert Wording	Select Formula	Crystal Generated WHERE Clause
Is equal to	=	=
Is not equal to	<>	<>
Is one of	In [ ]	Series of (field=value1 or field=value2 or ...)
Is not one of	Not (field in [ ])	Series of Not (field=value1 or field=value2 or ...)
Is less than	<	<
Is less than or equal to	<=	<=
Is greater than	>	>
Is greater than or equal to	>=	>=
Is between	Field in value1 to value2	(field>=value1 and field<=value2)
Is not between	Not (field in value1 to value2)	NOT (field>=value1 and field<=value2)
Starts with	Field starts with string	Field like string%
Does not start with	NOT (Field starts with string)	Field not like string%
Is like	Field like string	Field like string
Is not like	NOT (field like string)	Field not like string
Is null	IsNull()	Field IS NULL
Is not null	Not IsNull()	Field IS NOT NULL

**Table 3-1** Comparison Conditions

the database query. It also shows an alternative Oracle aggregation function that can be used in pass-through SQL.

The Crystal SQL column assumes that lower-level detail is suppressed or hidden and that grouping on the server is turned on. Otherwise, Crystal will not do any aggregation in the SQL query. If this column contains “Local”, it means that the aggregation is done locally and not pushed to the server. The Oracle Aggregation column displays an Oracle function that could be used in a SQL Command, view, or stored procedure, which is equivalent to the Crystal summary function. See Chapter 6 for detailed examples of Oracle substitutes for the Crystal summary functions.



### NOTE

*The percentile functions are new to Oracle 9i.*

In cases where the Crystal summary would be evaluated locally, performance gains can be made by substituting an Oracle function.

Crystal Summary	Available for Field Type			Crystal SQL	Oracle Aggregation
	Character	Numeric	Date		
Sum	No	Yes	No	SUM	SUM
Average	No	Yes	No	Local	AVG
Sample Variance	No	Yes	No	Local	VAR_SAMP
Sample Standard Deviation	No	Yes	No	Local	STDDEV_SAMP
Maximum	Yes	Yes	Yes	MAX	MAX
Minimum	Yes	Yes	Yes	MIN	MIN
Count	Yes	Yes	Yes	COUNT	COUNT
Distinct Count	Yes	Yes	Yes	Local	COUNT(DISTINCT())
Correlation with	No	Yes	No	Local	CORR
Covariance with	No	Yes	No	Local	COVAR_SAMP
Median	No	Yes	No	Local	PERCENTILE_CONT(0.5) PERCENTILE_DISC (0.5)
Mode	Yes	Yes	Yes	Local	See Chapter 6
Nth largest	Yes	Yes	Yes	Local	See Chapter 6
Nth smallest	Yes	Yes	Yes	Local	See Chapter 6
Nth most frequent	Yes	Yes	Yes	Local	See Chapter 6
Pth percentile	No	Yes	No	Local	PERCENT_RANK
Population Variance	No	Yes	No	Local	VAR_POP
Population Standard Deviation	No	Yes	No	Local	STDDEV_POP
Weighted Average With	No	Yes	No	Local	See Chapter 6

**Table 3-2** Summary Functions

Other available Oracle 8i functions include CUME\_DIST, DENSE\_RANK, FIRST\_VALUE, LAG, LAST\_VALUE, LEAD, NTILE, RANK, RATIO\_TO\_REPORT, and linear regression functions. FIRST and LAST are additional functions available in Oracle 9i.

## Working with Dates

As discussed in Chapter 2, Oracle has one datatype called DATE, which includes the date and the time down to the second. Oracle SQL accepts date literals in expressions

as well as numerous date functions. A date literal is a string representation of a date, such as '12-FEB-2002'. In Oracle tools, the format of the date literal must match the instance or session setting for the NLS\_DATE\_FORMAT parameter.

## Dates in the Selection Formula

Any selection criteria containing dates that is built using the Crystal Reports Select Expert will automatically create a selection formula that wraps the Crystal DateTime function around the date. This in turn will be translated into the SQL Query using the Oracle TO\_DATE function, as in the following example:

```
WHERE "EMPLOYEE"."BIRTH_DATE" <
      TO_DATE ('08-12-1972 00:00:00', 'DD-MM-YYYY HH24:MI:SS')
```

If you create the selection formula yourself, you should also use the Crystal DateTime function or the Crystal Date function. Using date literals will not be accepted in selection formulas.

## Date Literals in SQL Expressions

Date literals are allowed in Crystal Reports SQL Expressions. The Crystal Reports help states that when logging on to Oracle, the date format is changed to match the default Crystal Reports date format. This would seem to mean that Crystal is setting the session's NLS\_DATE\_FORMAT to match its own default DateTime format. However, testing shows that, no matter what the default Crystal DateTime format is set to, date literals of only a specific format are accepted. The accepted format is Year/Month/Day, where the year must be 4-digit; the month can be 2-digit, the 3-character abbreviation, or the whole month name; and the delimiter can be "/" or "-". This may be a bug. It is always safer to use the TO\_DATE Oracle function with a specific format string instead of date literals.

## DateTime Functions

Crystal Reports contains many date functions that can be used in formulas, but none of the Crystal Report date functions is translated to equivalent Oracle date functions in the SQL Query. If numerous or complex date manipulations are required, this could cause a slowdown in processing. To move this processing to the server, use SQL Expressions containing Oracle date functions.

The Oracle function TO\_DATE is used to convert a string to a date. This function is normally used with two parameters, the string that needs to be converted, and the string that contains the date format:

```
TO_DATE ('08-12-1972', 'DD-MM-YYYY')
```

If the format string is omitted, the date string must be in the same format as the NLS\_DATE\_FORMAT Oracle initialization parameter. A third parameter can be used to support globalization. See Oracle documentation for valid format strings.

Oracle has many Date/Time functions:

TO_CHAR	NUMTOYMINTERVAL
ADD_MONTHS	ROUND
CURRENT_DATE	SESSIONTIMEZONE
CURRENT_TIMESTAMP	SYS_EXTRACT_UTC
DBTIMEZONE	SYSDATE
EXTRACT	SYSTIMESTAMP
FROM_TZ	TO_DSINTERVAL
LAST_DAY	TO_TIMESTAMP
LOCALTIMESTAMP	TO_TIMESTAMP_TZ
MONTHS_BETWEEN	TO_YMINTERVAL
NEW_TIME	TRUNC
NEXT_DAY	TZ_OFFSET
NUMTODSINTERVAL	

Some of these functions use, or return, datatypes of timestamp or interval. Note that timestamp and interval types cannot be used directly by Crystal Reports. You can use any of the Oracle Date/Time functions in Crystal SQL Expressions even though some of them do not appear in the SQL Expression Editor lists.



### **NOTE**

*The functions listed in the preceding paragraph that use timestamp or interval datatypes are not available in Oracle 8i or previous versions.*

## **Working with Strings**

Crystal Reports contains many string functions that can be used in formulas, but none of the Crystal Report string functions is translated to equivalent Oracle character functions in the SQL Query. As with date and time data, if numerous or complex string manipulations are required, this could cause a slowdown in processing. To move this processing to the server, use SQL Expressions containing Oracle string functions.

Oracle character functions include the following:

TO_NUMBER	RPAD
CHR	RTRIM
CONCAT	SOUNDEX
INITCAP	SUBSTR
LOWER	TRANSLATE
LPAD	TRIM
LTRIM	UPPER
NLS_INITCAP	ASCII
NLS_LOWER	INSTR
NLSSORT	TRANSLATE
NLS_UPPER	LENGTH
REPLACE	

Most of these functions can be used with all Oracle character types, including CLOBs and LONGs. You can use any of the Oracle character functions in Crystal SQL Expressions even though some of them do not appear in the SQL Expression Editor.

---

## Working with Numbers

Crystal Reports contains many mathematical functions that can be used in formulas, but none of the Crystal Report math functions is translated to equivalent Oracle numeric functions in the SQL Query. Again, numerous or complex arithmetic manipulations could cause a slowdown in processing. Use SQL Expressions containing Oracle numeric functions to move this processing to the server.

Oracle numeric functions include the following:

TO_CHAR	LN
ABS	LOG
ACOS	MOD
ASIN	POWER
ATAN	ROUND
ATAN2	SIGN
BITAND	SIN
CEIL	SINH
COS	SQRT
COSH	TAN
EXP	TANH
FLOOR	TRUNC

You can use any of the Oracle numeric functions in Crystal SQL Expressions even though some of them do not appear in the SQL Expression Editor.

---

## Other Common Functions

Several Oracle built-in functions are commonly used in queries. Probably the two that are used most often are NVL and DECODE. You can use any of these functions in Crystal SQL Expressions even though some of them do not appear in the SQL Expression Editor.

### NVL

NVL is an extremely useful function that allows you to replace a null value with an appropriate substitute when needed. The following example will return 'Not Shipped' if the Ship\_Via field is null. NVL can also be used for numbers where a common replacement for null would be zero.

```
NVL ("ORDERS"."SHIP_VIA", 'Not Shipped')
```

### NVL2

NVL2 is a variation of NVL that allows you to return one value if the test value is null and a different value if the test value is not null:

```
NVL2 ("ORDERS"."SHIP_VIA", 'Ship Via is not null',
      'Ship Via is null')
```

### COALESCE

COALESCE is another variant of NVL. It takes a list of expressions and returns the first one that evaluates to a non-null value. The following example returns the Ship\_Date if it is not null. If the Ship\_Date is null and the Required\_Date is not null, it returns the Required\_Date. If both the Ship\_Date and Required\_Date are null, it returns the Order\_Date. If all three are null, it returns null.

```
COALESCE ("ORDERS"."SHIP_DATE", "ORDERS"."REQUIRED_DATE",
         "ORDERS"."ORDER_DATE")
```



#### **NOTE**

---

*The COALESCE function is new to Oracle 9i.*

## DECODE

The DECODE function takes a test value and a list of pairs of values. It returns the second value in the pair where the test value matches the first value in the pair. In the example that follows, if Employee\_ID=1, then 'Joe' will be returned; if Employee\_ID=2, then 'Jane' will be returned; if Employee\_ID is not 1 or 2, then 'Everyone else' will be returned:

```
DECODE("ORDERS"."EMPLOYEE_ID", 1, 'Joe', 2, 'Jane',
      'Everyone else')
```

## CASE

The Oracle CASE keyword is not really a function but an expression that can be used in a SQL query. In its simple format, it is similar to the DECODE function:

```
CASE "ORDERS"."EMPLOYEE_ID"
  WHEN 1 THEN 'Joe'
  WHEN 2 THEN 'Jane'
  ELSE 'Everybody else'
END
```

It can also be used in a more complex manner where each condition is not relative to the same field, but completely independent. This called a searched case statement.

```
CASE
  WHEN "ORDERS"."ORDER_AMOUNT">10000 THEN 'Big Order'
  WHEN "ORDERS"."CUSTOMER_ID"=5 THEN 'Big Customer'
  ELSE 'Normal Order'
END
```



### NOTE

---

*Simple case expressions were available in Oracle 8i, but searched case expressions are new to Oracle 9i.*

## GREATEST

The GREATEST function returns the largest value from a list of values. If items in the list are not of the same type, they are converted to the type of the first value. In the following example, whichever date is the latest will be returned. GREATEST is not the same as MAX. MAX is an aggregation function that returns the greatest

value in a column over the entire group, whereas GREATEST is a single row function.

```
GREATEST ("ORDERS"."ORDER_DATE", "ORDERS"."REQUIRED_DATE",
          "ORDERS"."SHIP_DATE")
```

## LEAST

The LEAST function returns the smallest value from a list of values. As with GREATEST, if items in the list are not of the same type, they are converted to the type of the first value. In the example that follows, whichever date is the earliest will be returned. LEAST is not equivalent to MIN:

```
LEAST ("ORDERS"."ORDER_DATE", "ORDERS"."REQUIRED_DATE",
       "ORDERS"."SHIP_DATE")
```

---

## Pseudo Columns

Pseudo columns are values that Oracle maintains. You can select them as if they were regular columns.

## ROWNUM

The rownum pseudo column is a number showing in what order Oracle selected the rows in the result set. The rownum value can vary depending on ORDER BY clauses and other conditions. It does not represent a constant value for each row in a table. The rownum pseudo column is often used to limit the result set for sampling purposes or to create Top-N type queries.

To limit the number of rows returned, create a SQL Expression whose value is rownum and then use that SQL Expression in the select formula. Be aware that the rownum determination is happening before the sort so this will not return the top 15 Employee\_IDs, just the first 14 rows retrieved from Orders sorted by Employee\_ID. The sort happens after the retrieval:

```
SELECT "ORDERS"."EMPLOYEE_ID", "ORDERS"."ORDER_AMOUNT",
       "ORDERS"."ORDER_ID", (rownum)
FROM "XTREME"."ORDERS" "ORDERS"
WHERE (ROWNUM) < 15
ORDER BY "ORDERS"."EMPLOYEE_ID"
```

To create a Top-N query using rownum, you must use a SQL Command. For instance, if your SQL Command contained the following statement, you would get the smallest 14 orders by Order\_amount:

```
SELECT * FROM
  (SELECT "ORDERS"."EMPLOYEE_ID",
         "ORDERS"."ORDER_AMOUNT", "ORDERS"."ORDER_ID"
   FROM "XTREME"."ORDERS" "ORDERS"
   ORDER BY "ORDERS"."ORDER_AMOUNT")
WHERE ROWNUM<15
```

## ROWID

The ROWID pseudo column is the physical row address. Even this value should not be considered constant for a row. It can change in cases of row migration or table restructuring.

---

## Subqueries

Subqueries are queries that are nested inside another query. Subqueries can appear in two places in SELECT statements. A subquery in the FROM clause is also called an inline view. This example shows an inline view:

```
SELECT "EMPLOYEE_ID",
       "ORDERS"."ORDER_AMOUNT",
       "ORDERS"."ORDER_ID",
       "ORDERS3"."AVG_ORDER"
  FROM "XTREME"."ORDERS" "ORDERS"
  JOIN
    (SELECT "ORDERS2"."EMPLOYEE_ID",
           AVG("ORDERS2"."ORDER_AMOUNT") AVG_ORDER
     FROM "XTREME"."ORDERS" "ORDERS2"
     GROUP BY "ORDERS2"."EMPLOYEE_ID") "ORDERS3"
  USING ("EMPLOYEE_ID")
  ORDER BY "ORDERS"."ORDER_AMOUNT"
```

A subquery in the WHERE clause is called a nested subquery. Subqueries can be nested inside other subqueries. This example shows an uncorrelated nested subquery:

```
SELECT "ORDERS"."EMPLOYEE_ID",
       "ORDERS"."ORDER_AMOUNT", "ORDERS"."ORDER_ID"
```

```

FROM "XTREME"."ORDERS" "ORDERS"
WHERE "ORDERS"."ORDER_AMOUNT">
      (SELECT AVG("ORDERS2"."ORDER_AMOUNT")
       FROM "XTREME"."ORDERS" "ORDERS2")
ORDER BY "ORDERS"."ORDER_AMOUNT"

```

If a nested subquery contains a reference to a field in the main query in its WHERE clause, it is called a correlated nested subquery. This example shows a correlated nested subquery:

```

SELECT "ORDERS"."EMPLOYEE_ID", "ORDERS"."ORDER_AMOUNT",
       "ORDERS"."ORDER_ID"
FROM "XTREME"."ORDERS" "ORDERS"
WHERE "ORDERS"."ORDER_AMOUNT">
      (SELECT AVG("ORDERS2"."ORDER_AMOUNT")
       FROM "XTREME"."ORDERS" "ORDERS2"
       WHERE "ORDERS2"."EMPLOYEE_ID"="ORDERS"."EMPLOYEE_ID")
ORDER BY "ORDERS"."ORDER_AMOUNT"

```

This chapter covered basic Oracle SELECT statement construction. How Crystal Reports constructs SQL statements to send to Oracle was demonstrated, as were statement options beyond the simple syntax used by Crystal. The next chapter will describe more complex SELECT statement options.