THE LANGUAGE of SQL

SECOND EDITION

LARRY ROCKOFF

The Language of SQL

Second Edition

Larry Rockoff

Addison-Wesley

Hoboken, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Because we're now specifying the percent (%) wildcard only after the word LOVE, we get back only movies that begin with LOVE. Similarly, if we issue:

```
SELECT
MovieTitle AS 'Movie'
FROM Movies
WHERE MovieTitle LIKE '%LOVE'
```

we get only this data:

Movie
Down with Love

This is because we have now specified that the phrase must *end* with the word LOVE.

One might ask how to arrange the wildcards to see only movies that contain the word LOVE in the middle of the title, without seeing movies where LOVE is at the beginning or end. The solution is to specify:

```
SELECT
MovieTitle AS 'Movie'
FROM Movies
WHERE MovieTitle LIKE '% LOVE %'
```

Notice that a space has been inserted between the word LOVE and the percent (%) wildcards on either side. This ensures that there is at least one space on both sides of the word. The data brought back from this statement is:

Movie
Everyone Says I Love You

Wildcards

The percent (%) sign is the most common wildcard used with the LIKE operator, but there are a few other possibilities. These include the underscore character (_), a *characterlist* enclosed in square brackets, and a caret symbol (^) plus a *characterlist* enclosed in square brackets. The following table lists these wildcards and their meanings:

Wildcard	Meaning
%	any characters (can be zero characters)
_	exactly one character (can be any character)
[*characterlist*]	exactly one character in the *characterlist*
[^*characterlist*]	exactly one character *not* in the *characterlist*

We'll use the following Actors table to illustrate statements for these wildcards.

ActorID	FirstName	LastName
1	Cary	Grant
2	Mary	Steenburgen
3	Jon	Voight
4	Dustin	Hoffman
5	John	Wayne
6	Gary	Cooper

Here's an illustration of how the underscore (_) wildcard character can be used:

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE '_ARY'
```

The output of this SELECT is:

FirstName	LastName
Cary	Grant
Mary	Steenburgen
Gary	Cooper

This statement retrieves these three actors because all have a first name consisting of exactly one character followed by the phrase ARY.

Likewise, if we issue this statement:

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE 'J_N'
```

it produces:

FirstName	LastName
Jon	Voight

The actor John Wayne is not selected since John doesn't fit the J_N pattern. An underscore stands for only one character.

The final wildcards we'll discuss, [*characterlist*] and [^*characterlist*], enable you to specify multiple wildcard values in a single position.

> **Database Differences: MySQL and Oracle**
>
> The [*characterlist*] and [^*characterlist*] wildcards are not available in MySQL or Oracle.

The following illustrates the [*characterlist*] wildcard:

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE '[CM]ARY'
```

This retrieves any rows where FirstName begins with a C or M and ends with ARY. The result is:

FirstName	LastName
Cary	Grant
Mary	Steenburgen

The following illustrates the [^*characterlist*] wildcard:

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE '[^CM]ARY'
```

This selects any rows where FirstName does not begin with a C or M and ends with ARY. The result is:

FirstName	LastName
Gary	Cooper

Looking Ahead

This chapter introduced the topic of how to apply selection criteria to queries. A number of basic operators, such as equals and greater than, were introduced. The ability to specify these types of basic selection criteria goes a long way toward making the SELECT statement truly useful. We also covered the related topic of limiting the number of rows returned in a query. The ability to limit rows in combination with an ORDER BY clause allows for a useful Top N type of data selection.

We concluded the chapter with a study of matching words or phrases via a specified pattern. Matching by patterns is a significant and widely used function of SQL. Any time you enter a word in a search box and attempt to retrieve all entities containing that word, you are utilizing pattern matching.

In our next chapter, "Boolean Logic," we'll greatly enhance our selection criteria capabilities by introducing a number of new keywords that add sophisticated logic to the WHERE clause. At present, we can do such things as select all customers from the state of New York. In the real world, however, much more is typically required. Boolean logic will allow us to formulate a query that will select customers who are in New York or California but not in Los Angeles or Albuquerque.

This page intentionally left blank

7

Boolean Logic

Keywords Introduced

AND · OR · NOT · BETWEEN · IN · IS NULL

We introduced the concept of selection criteria in the previous chapter, but only in its simplest form. We'll now expand on that concept to greatly enhance our ability to specify the rows returned from a SELECT. This is where the pure logic of SQL comes into play. In this chapter, we'll introduce a number of operators that will allow you to create complex logical expressions.

Given these new capabilities, if a user comes to you and requests a list of all female customers who live in zip codes 60601 through 62999 but excluding anyone under the age of 30 or who doesn't have an email address, that will be something you can provide.

Complex Logical Conditions

The WHERE clause introduced in the previous chapter used only simple selection criteria. We saw clauses such as:

```
WHERE QuantityPurchased = 5
```

The condition expressed in this WHERE clause is quite basic. It specifies merely to return all rows where the QuantityPurchased column has a value of 5. In the real world, the selection of data is often far from this straightforward. Accordingly, let's now turn our attention to methods of specifying some more complex logical conditions in selection criteria.

The ability to devise complex logical conditions is sometimes called *Boolean logic*. This term, taken from mathematics, refers to the ability to formulate complex conditions that are evaluated as either true or false. In the aforementioned example, the condition QuantityPurchased = 5 is evaluated as either true or false for each row in the table. Obviously, we want to see only rows where the condition is evaluated as true.