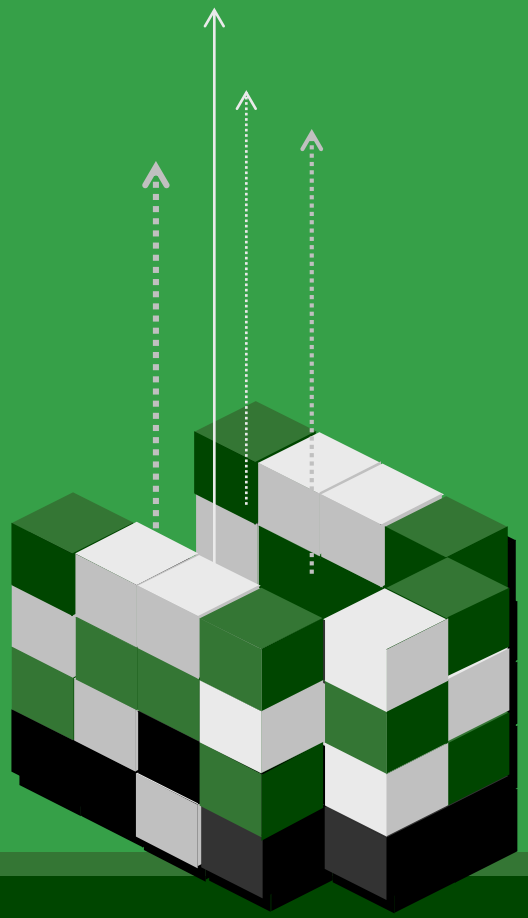
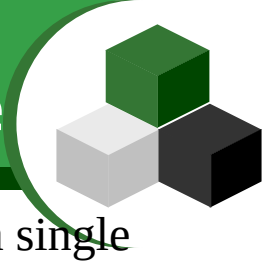
chaper 2

Introduction to Structured Query Language



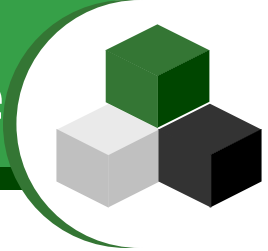
5. SQL Enhancements for Querying a Single Table



We started our discussion of SQL queries with SQL statements for processing a single table, and now we will add an additional SQL feature to those queries. As we proceed, you will begin to see how powerful SQL can be for querying databases and for creating information from existing data.

By the way The SQL results shown in this chapter were generated using Microsoft SQL Server 2008 R2. Query results from other DBMS products will be similar, but may vary a bit.

5. SQL Enhancements for Querying a Single Table



- **Sorting the SQL Query Results**

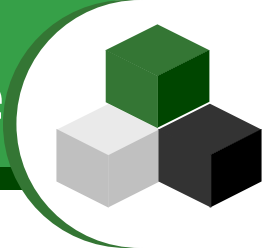
The order of the rows produced by an SQL statement is arbitrary and determined by programs in the bowels of each DBMS. If you want the DBMS to display the rows in a particular order, you can use the SQL ORDER BY clause. For example, the SQL statement:

```
/* *** SQL-Query-CH02-10 *** */
SELECT        *
FROM  ORDER_ITEM
ORDER  BY   OrderNumber;
```

will generate the following results:

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	1000	201000	1	300.00	300.00
2	1000	202000	1	130.00	130.00
3	2000	101100	4	50.00	200.00
4	2000	101200	2	50.00	100.00
5	3000	101200	1	50.00	50.00
6	3000	101100	2	50.00	100.00
7	3000	100200	1	300.00	300.00

5. SQL Enhancements for Querying a Single Table



- **Sorting the SQL Query Results**

  We can sort by two columns by adding a second column name. For example, to sort first by OrderNumber and then by Price within OrderNumber, we use the following SQL query:

```
/* *** SQL-Query-CH02-11 *** */
SELECT        *
FROM  ORDER_ITEM
ORDER  BY   OrderNumber,  Price;
```

The result for this query is:

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	1000	202000	1	130.00	130.00
2	1000	201000	1	300.00	300.00
3	2000	101100	4	50.00	200.00
4	2000	101200	2	50.00	100.00
5	3000	101200	1	50.00	50.00
6	3000	101100	2	50.00	100.00
7	3000	100200	1	300.00	300.00

5. SQL Enhancements for Querying a Single Table

➢Sorting the SQL Query Results

If we want to sort the data by Price and then by OrderNumber, we would simply reverse the order of those columns in the ORDER BY clause as follows:

```
/* *** SQL-Query-CH02-12 *** */
SELECT        *
FROM  ORDER_ITEM
ORDER  BY   Price,  OrderNumber;
```

with the results:

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	2000	101100	4	50.00	200.00
2	2000	101200	2	50.00	100.00
3	3000	101200	1	50.00	50.00
4	3000	101100	2	50.00	100.00
5	1000	202000	1	130.00	130.00
6	1000	201000	1	300.00	300.00
7	3000	100200	1	300.00	300.00

By the way Note to Microsoft Access users: Unlike the SQL Server output shown here, Microsoft Access displays dollar signs in the output of currency data.

5. SQL Enhancements for Querying a Single Table

➢Sorting the SQL Query Results

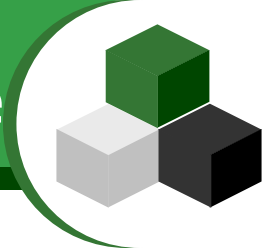
By default, rows are sorted in ascending order. To sort in descending order, add the SQL DESC keyword after the column name. Thus, to sort first by Price in descending order and then by OrderNumber in ascending order, we use the SQL query:

```
/* *** SQL-Query-CH02-13 *** */
SELECT        *
FROM  ORDER_ITEM
ORDER  BY   Price  DESC,  OrderNumber  ASC;
```

The result is:

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	1000	201000	1	300.00	300.00
2	3000	100200	1	300.00	300.00
3	1000	202000	1	130.00	130.00
4	2000	101100	4	50.00	200.00
5	2000	101200	2	50.00	100.00
6	3000	101200	1	50.00	50.00
7	3000	101100	2	50.00	100.00

5. SQL Enhancements for Querying a Single Table



➢Sorting the SQL Query Results

Because the default order is ascending, it is not necessary to specify ASC in the last SQL statement. Thus, the following SQL statement is equivalent to the previous SQL query:

```
/* *** SQL-Query-CH02-14 *** */
SELECT        *
FROM  ORDER_ITEM
ORDER  BY    Price  DESC,  OrderNumber;
```

and produces the same results:

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	1000	201000	1	300.00	300.00
2	3000	100200	1	300.00	300.00
3	1000	202000	1	130.00	130.00
4	2000	101100	4	50.00	200.00
5	2000	101200	2	50.00	100.00
6	3000	101200	1	50.00	50.00
7	3000	101100	2	50.00	100.00

5. SQL Enhancements for Querying a Single Table

➢SQL WHERE Clause Options

SQL includes a number of SQL WHERE clause options that greatly expand SQL's power and utility. In this section, we consider three options: compound clauses, ranges, and wildcards.

Compound WHERE Clauses

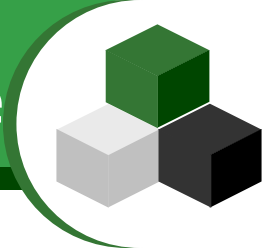
SQL WHERE clauses can include multiple conditions by using the SQL AND, OR, IN, and NOT IN operators. For example, to find all of the rows in SKU_DATA that have a Department named Water Sports and a Buyer named Nancy Meyers, we can use the SQL AND operator in our query code:

```
/* *** SQL-Query-CH02-15 *** */
SELECT        *
FROM  SKU_DATA
WHERE         Department='Water  Sports'
AND   Buyer='Nancy  Meyers';
```

The results of this query are:

	SKU	SKU_Description	Department	Buyer
1	101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
2	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers

5. SQL Enhancements for Querying a Single Table



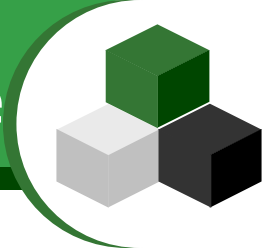
Similarly, to find all of the rows of SKU_DATA for either the Camping or Climbing departments, we can use the SQL OR operator in the SQL query:

```
/* *** SQL-Query-CH02-16 *** */
SELECT        *
FROM  SKU_DATA
WHERE         Department='Camping'
OR      Department='Climbing';
```

which gives us the following results:

	SKU	SKU_Description	Department	Buyer
1	201000	Half-dome Tent	Camping	Cindy Lo
2	202000	Half-dome Tent Vestibule	Camping	Cindy Lo
3	301000	Light Fly Climbing Harness	Climbing	Jerry Martin
4	302000	Locking Carabiner, Oval	Climbing	Jerry Martin

5. SQL Enhancements for Querying a Single Table



Three or more AND and OR conditions can be combined, but in such cases the SQL IN operator and the SQL NOT IN operator are easier to use. For example, suppose we want to obtain all of the rows in SKU_DATA for buyers Nancy Meyers, Cindy Lo, and Jerry Martin. We could construct a WHERE clause with two ANDs, but an easier way to do this is to use the IN operator, as illustrated in the SQL query:

```
/* *** SQL-Query-CH02-17 *** */
SELECT      *
FROM  SKU_DATA
WHERE       Buyer  IN  ('Nancy  Meyers',  'Cindy  Lo',  'Jerry
Martin');
```

In this format, a set of values is enclosed in parentheses. A row is selected if Buyer is equal to any one of the values provided. The result is:

	SKU	SKU_Description	Department	Buyer
1	101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
2	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
3	201000	Half-dome Tent	Camping	Cindy Lo
4	202000	Half-dome Tent Vestibule	Camping	Cindy Lo
5	301000	Light Fly Climbing Harness	Climbing	Jerry Martin
6	302000	Locking Carabiner, Oval	Climbing	Jerry Martin

5. SQL Enhancements for Querying a Single Table

Similarly, if we want to find rows of SKU_DATA for which the buyer is someone other than Nancy Meyers, Cindy Lo, or Jerry Martin, we would use the SQL query:

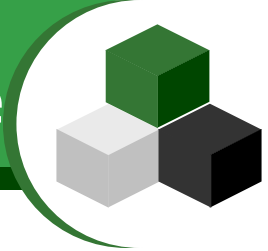
```
/* *** SQL-Query-CH02-18 *** */
SELECT      *
FROM  SKU_DATA
WHERE       Buyer  NOT  IN  ('Nancy  Meyers',  'Cindy  Lo',
'Jerry  Martin');
```

The result is:

	SKU	SKU_Description	Department	Buyer
1	100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen

Observe an important difference between IN and NOT IN. A row qualifies for an IN condition if the column is equal to any of the values in the parentheses. However, a row qualifies for a NOT IN condition if it is not equal to all of the items in the parentheses.

5. SQL Enhancements for Querying a Single Table



Ranges in SQL WHERE Clauses

SQL WHERE clauses can specify ranges of data values by using the SQL BETWEEN keyword. For example, the following SQL statement:

```
/* *** SQL-Query-CH02-19 *** */
SELECT        *
FROM  ORDER_ITEM
WHERE         ExtendedPrice  BETWEEN  100  AND  200;
```

will produce the following results:

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	2000	101100	4	50.00	200.00
2	3000	101100	2	50.00	100.00
3	2000	101200	2	50.00	100.00
4	1000	202000	1	130.00	130.00

5. SQL Enhancements for Querying a Single Table

Ranges in SQL WHERE Clauses

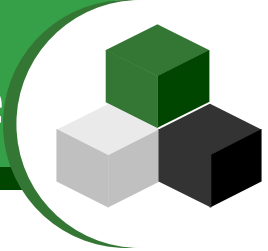
Notice that both the ends of the range, 100 and 200, are included in the resulting table. The preceding SQL statement is equivalent to the SQL query:

```
/* *** SQL-Query-CH02-20 *** */
SELECT        *
FROM  ORDER_ITEM
WHERE         ExtendedPrice  >=  100
AND    ExtendedPrice  <=  200;
```

And which, of course, produces identical results:

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	2000	101100	4	50.00	200.00
2	3000	101100	2	50.00	100.00
3	2000	101200	2	50.00	100.00
4	1000	202000	1	130.00	130.00

5. SQL Enhancements for Querying a Single Table



Wildcards in SQL WHERE Clauses

The SQL LIKE keyword can be used in SQL WHERE clauses to specify matches on portions of column values. For example, suppose we want to find the rows in the SKU_DATA table for all buyers whose first name is Pete. To find such rows, we use the SQL keyword LIKE with the SQL percent sign (%) wildcard character, as shown in the SQL query:

```
/* *** SQL-Query-CH02-21 *** */
SELECT      *
FROM  SKU_DATA
WHERE       Buyer  LIKE  'Pete%';
```

When used as an SQL wildcard character, the percent symbol (%) stands for any sequence of characters. When used with the SQL LIKE keyword, the character string 'Pete%' means any sequence of characters that start with the letters Pete. The result of this query is:

	SKU	SKU_Description	Department	Buyer
1	100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen

5. SQL Enhancements for Querying a Single Table

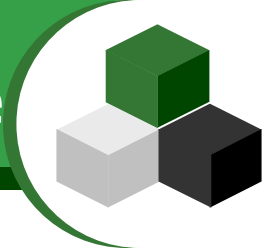
Does Not Work With MS Access ANSI-89 SQL

Microsoft Access ANSI-89 SQL uses wildcards, but not the SQL-92 standard wildcards. Microsoft Access uses the Microsoft Access asterisk (*) wildcard character instead of a percent sign to represent multiple characters.

Solution: Use the Microsoft Access asterisk (*) wildcard in place of the SQL-92 percent sign (%) wildcard in Microsoft Access ANSI-89 SQL statements. Thus, the preceding SQL query would be written as follows for Microsoft Access:

```
/*  ***  SQL-Query-CH02-21-Access  ***          */
SELECT     *
FROM       SKU_DATA
WHERE      Buyer  LIKE  'Pete*';
```

5. SQL Enhancements for Querying a Single Table



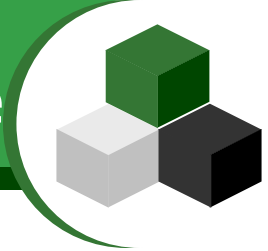
Suppose we want to find the rows in SKU_DATA for which the SKU_Description includes the word Tent somewhere in the description. Because the word Tent could be at the front, the end, or in the middle, we need to place a wildcard on both ends of the LIKE phrase, as follows:

```
/* *** SQL-Query-CH02-22 *** */
SELECT        *
FROM  SKU_DATA
WHERE         Buyer  LIKE  '%Tent%';
```

This query will find rows in which the word Tent occurs in any place in the SKU_Description. The result is:

	SKU	SKU_Description	Department	Buyer
1	201000	Half-dome Tent	Camping	Cindy Lo
2	202000	Half-dome Tent Vestibule	Camping	Cindy Lo

5. SQL Enhancements for Querying a Single Table



Sometimes we need to search for a particular value in a particular location in the column. For example, assume SKU values are coded such that a 2 in the third position from the right has some particular significance, maybe it means that the product is a variation of another product. For whatever reason, assume that we need to find all SKUs that have a 2 in the third column from the right. Suppose we try the SQL query:

```
/* *** SQL-Query-CH02-23 *** */
SELECT      *
FROM  SKU_DATA
WHERE       SKU  LIKE  '%2%';
```

The result is:

	SKU	SKU_Description	Department	Buyer
1	100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
2	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
3	201000	Half-dome Tent	Camping	Cindy Lo
4	202000	Half-dome Tent Vestibule	Camping	Cindy Lo
5	302000	Locking Carabiner, Oval	Climbing	Jerry Martin

5. SQL Enhancements for Querying a Single Table

This is not what we wanted. We mistakenly retrieved all rows that had a 2 in any position in the value of SKU. To find the products we want, we cannot use the SQL wildcard character %. Instead, we must use the SQL underscore (_) wildcard character, which represents a single, unspecified character in a specific position. The following SQL statement will find all SKU_DATA rows with a value of 2 in the third position from the right:

```
/* *** SQL-Query-CH02-24 *** */
SELECT        *
FROM  SKU_DATA
WHERE         SKU  LIKE '%2__';
```

Observe that there are two underscores in this SQL query— one for the first position on the right and another for the second position on the right. This query gives us the result that we want:

	SKU	SKU_Description	Department	Buyer
1	100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
2	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers

5. SQL Enhancements for Querying a Single Table

Does Not Work With MS Access ANSI-89 SQL

Microsoft Access ANSI-89 SQL uses wildcards, but not the SQL-92 standard wildcards. Microsoft Access uses the Microsoft Access question mark (?) wildcard character instead of an underscore (_) to represent a single character.

Solution: Use the Microsoft Access question mark (?) wildcard in place of the SQL-92 underscore (_) wildcard in Microsoft Access ANSI-89 SQL statements. Thus, the preceding SQL query would be written as follows for Microsoft Access:

```
/* *** SQL-Query-CH02-24-Access *** */
SELECT*
FROM   SKU_DATA
WHERE SKU  LIKE  '*2??';
```

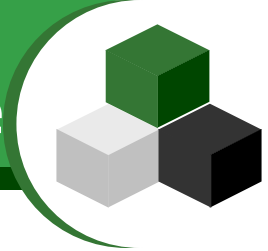
Furthermore, Microsoft Access can sometimes be fussy about stored trailing spaces in a text field. You may have problems with a WHERE clause like this:

```
WHERE SKU  LIKE  '10?200';
```

Solution: Use a trailing asterisk (*), which allows for the trailing spaces:

```
WHERE SKU  LIKE  '10?200*';
```

5. SQL Enhancements for Querying a Single Table



By the way The SQL wildcard percent sign (%) and underscore (_) characters are specified in the SQL-92 standard. They are accepted by all DBMS products except Microsoft Access. So, why does Microsoft Access use the asterisk (*) character instead of the percent sign (%) and the question mark (?) instead of the underscore? This difference probably exists because the designers of Microsoft Access chose to use the same wildcard characters that Microsoft was already using in the Microsoft MS-DOS operating system.

5. SQL Enhancements for Querying a Single Table

➢ Combining the SQL WHERE Clause and the SQL ORDER BY Clause

If we want to sort the results generated by these enhanced SQL WHERE clauses, we simply combine the SQL ORDER BY clause with the WHERE clause. This is illustrated by the following SQL query:

```
/* *** SQL-Query-CH02-25 *** */
SELECT        *
FROM  ORDER_ITEM
WHERE         ExtendedPrice  BETWEEN  100  AND  200
ORDER  BY     OrderNumber  DESC;
```

which will produce the following result:

	OrderNumber	SKU	Quantity	Price	ExtendedPrice
1	3000	101100	2	50.00	100.00
2	2000	101200	2	50.00	100.00
3	2000	101100	4	50.00	200.00
4	1000	202000	1	130.00	130.00

LOGO

Thank You!