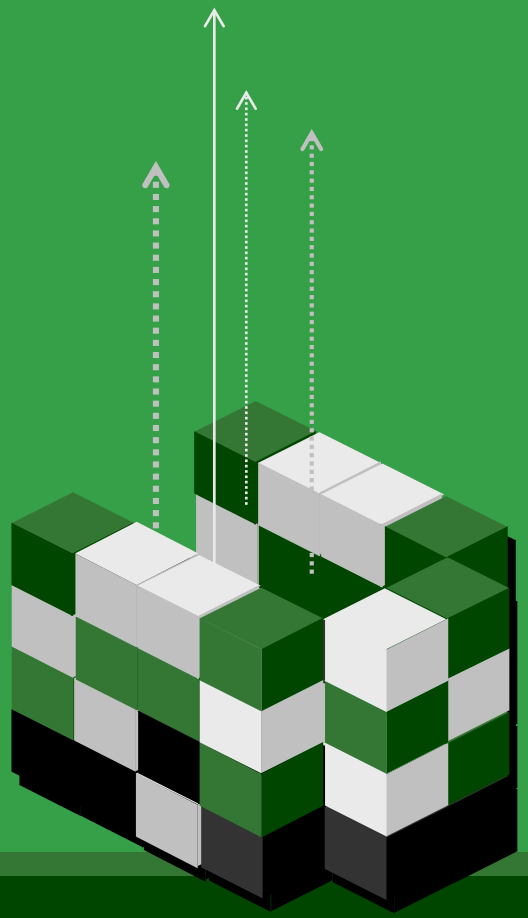
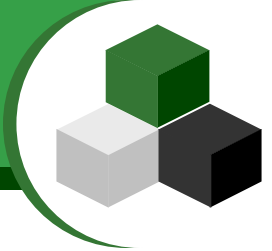
chaper 2

Introduction to Structured Query Language

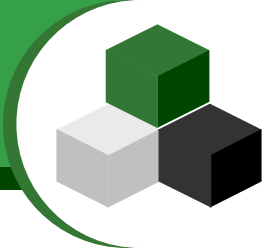


chaper 2 Objectives

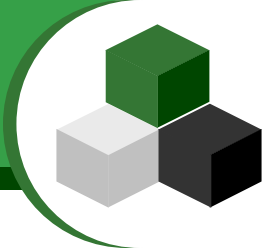


- To understand the use of extracted data sets.
- To understand the use of ad-hoc queries.
- To understand the history and significance of Structured Query Language (SQL).
- To understand the SQL SELECT/FROM/WHERE framework as the basis for database queries.
- To create SQL queries to retrieve data from a single table
- To create SQL queries that use the SQL SELECT, FROM, WHERE, ORDER BY, GROUP BY, and HAVING clauses
- To create SQL queries that use the SQL DISTINCT, AND, OR, NOT, BETWEEN, LIKE, and IN keywords
- To create SQL queries that use the SQL built-in functions of SUM, COUNT, MIN, MAX, and AVG with and without the SQL GROUP BY clause
- To create SQL queries that retrieve data from a single table while restricting the data based upon data in another table (subquery)
- To create SQL queries that retrieve data from multiple tables using the SQL JOIN operation

Contents



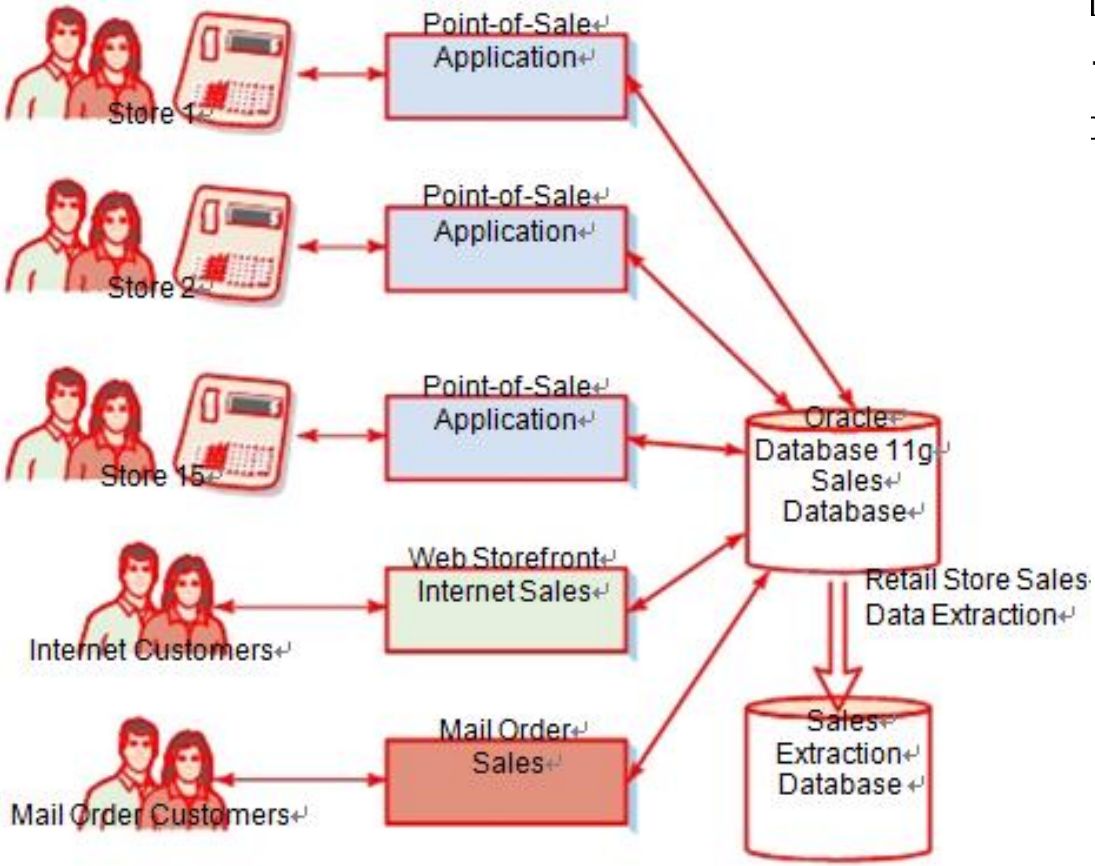
Contents



1. Cape Codd Outdoor Sports

For our work in this chapter, we will use data from Cape Codd Outdoor Sports (although based on a real outdoor retail equipment vendor, Cape Codd Outdoor Sports is a fictitious company). Cape Codd sells recreational outdoor equipment in 15 retail stores across the United States and Canada. It also sells merchandise over the Internet from a Web storefront application and via mail order. All retail sales are recorded in a sales database manage -1.

Figu on



1. Cape Codd Outdoor Sports

➢The Retail Sales Data Extraction

Cape Codd’s marketing department wants to perform an analysis of in-store sales. Accordingly, marketing analysts ask the IT department to extract retail sales data from the operational database. To perform the marketing study, they do not need all of the order data. They want just the tables and columns shown in Figure 2-2. The data types for the columns in the tables is shown in Figure 2-3.

As shown in Figures 2-2 and Figures 2-3, three tables are needed: RETAIL_ORDER, ORDER_ITEM, and SKU_DATA. The RETAIL_ORDER table has data about each retail sales order, the ORDER_ITEM table has data about each item in an order, and the SKU_DATA table has data about each stock-keeping unit (SKU). SKU is a unique identifier for each particular item that Cape Codd sells. The data stored in the tables is shown in Figure 2-4.

1. Cape Codd Outdoor Sports

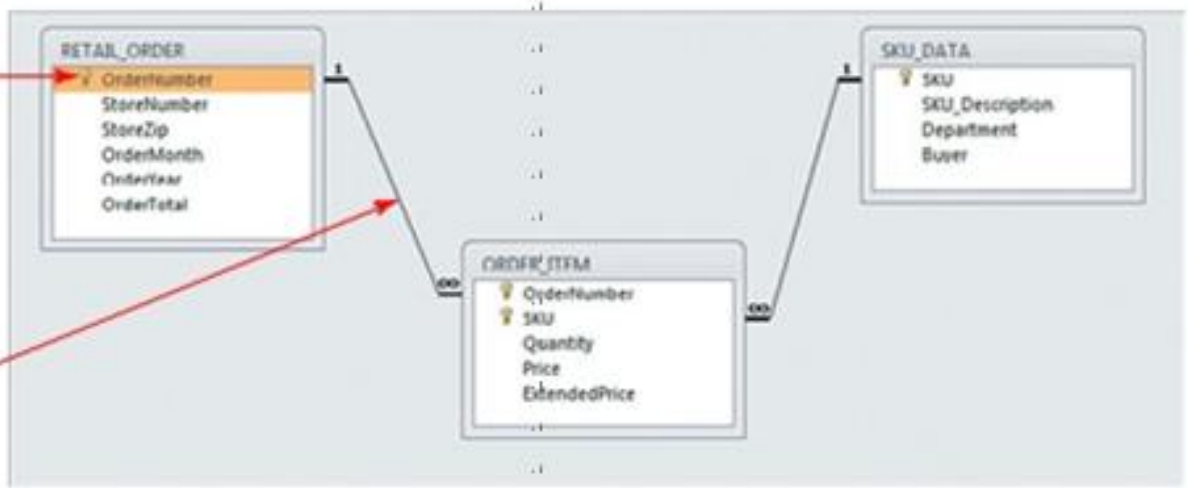
➢**The Retail Sales Data Extraction**

Figure 2-2

Cape Codd Extracted Retail Sales Data Database Tables and Relationships

The RETAIL_ORDER table—the key symbol shows the primary key

The relationship between RETAIL_ORDER and ORDER_ITEM—the number 1 and the infinity symbol indicate that one retail order may be linked to many order items by OrderNumber



1. Cape Codd Outdoor Sports

➢**The Retail Sales Data Extraction**

Figure 2-3 Cape Codd Extracted Retail Sales Data Format

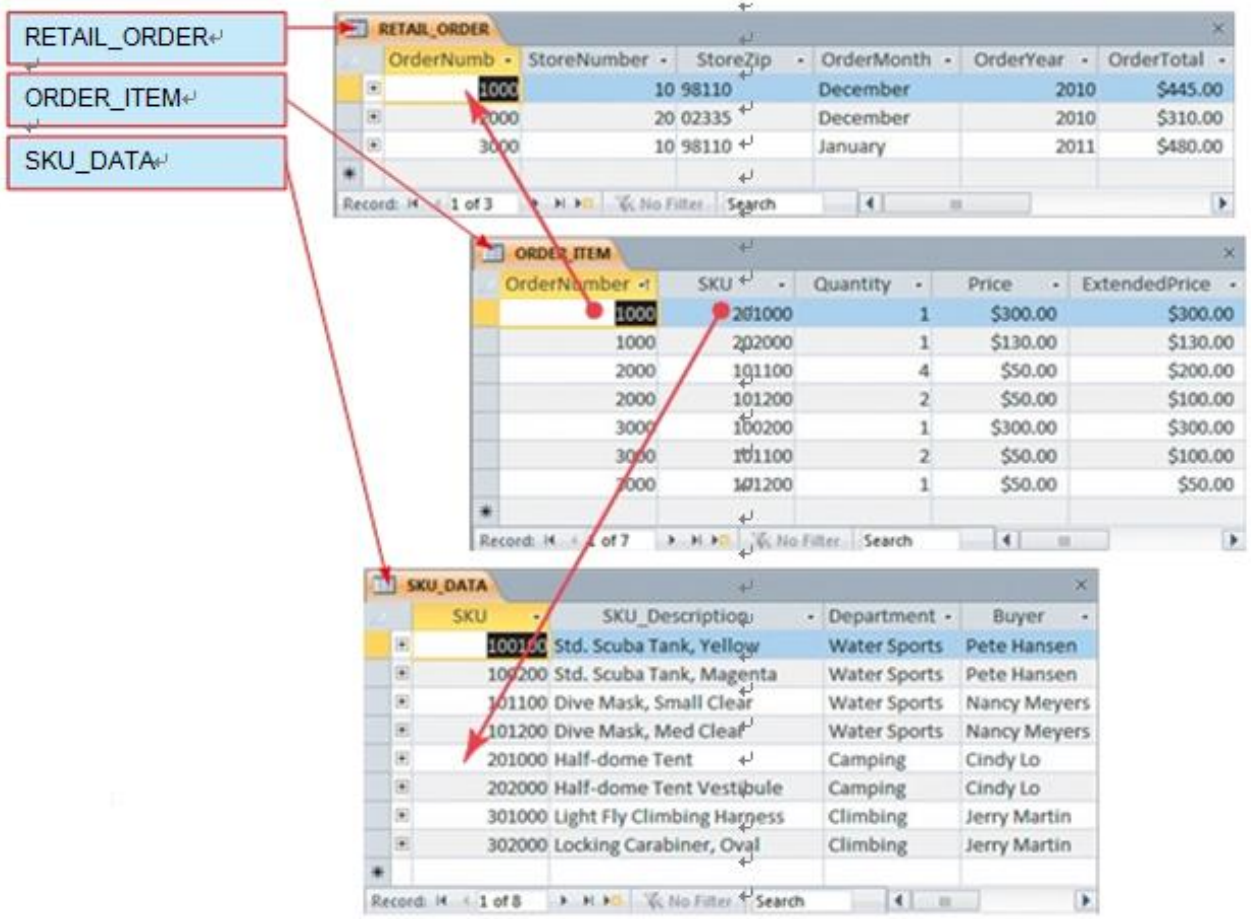
Table	Column	Date Type
RETAIL_ORDER	OrderNumber	Integer
	StoreNumber	Integer
	StoreZip	Character (9)
	OrderMonth	Character (12)
	OrderYear	Integer
	OrderTotal	Currency
ORDER_ITEM	OrderNumber	Integer
	SKU	Integer
	Quantity	Integer
	Price	Currency
	ExtendedPrice	Currency
SKU_DATA	SKU	Integer
	SKU_Description	Character (35)
	Department	Character (30)
	Buyer	Character (30)

1. Cape Codd Outdoor Sports

➢**The Retail Sales Data Extraction**

Figure 2-4

Sample Data in the Cape Codd Extracted Retail Sales Database



1. Cape Codd Outdoor Sports

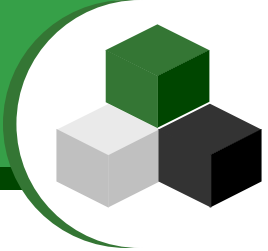
➢RETAIL_ORDER Data

As shown in Figures 2-2 and 2-3, the RETAIL_ORDER table has columns for OrderNumber, StoreNumber, StoreZip (the zip code of the store selling the order), OrderMonth, OrderYear, and OrderTotal:

RETAIL_ORDER (OrderNumber, StoreNumber, StoreZip, OrderMonth, OrderYear, OrderTotal)

Sample data for RETAIL_ORDER is shown in Figure 2-5. This extract only includes data for retail store sales, and operational data for other types of sales (and returns and other salesrelated transactions) are not copied during the extraction process. Further, the data extraction process selects only a few columns of the operational data—the Point of Sale (POS) and other sales applications process far more data than that shown here. The operational database also stores the data in a different format. For example, the order data in the an Oracle Database 11g DBMS operational database contains a column named OrderDate that stores the data in the date format MM/DD/YYYY (e.g., 10/22/2010 for October 22, 2010). The extraction program used to populate the retail sales extracted data database converts OrderDate into two separate values of OrderMonth and OrderYear. This is done because this is the data format that marketing wants. Such filtering and data transformation are typical of a data extraction process.

1. Cape Codd Outdoor Sports



➢ORDER_ITEM Data

As shown in Figures 2-2 and 2-3, the ORDER_ITEM table has columns for OrderNumber, SKU, Quantity, Price, and ExtendedPrice (which equals Quantity × Price):

ORDER_ITEM (OrderNumber, SKU, Quantity, Price, ExtendedPrice)

Thus, the ORDER_ITEM table stores an extract of the items purchased in each order. There is one row in the table for each item in an order, and this item is identified by its SKU. To understand this table, think about a sales receipt you get from a retail store. That receipt has data for one order. It includes basic order data such as the date and order total, and it has one line for each item you purchase. The rows in the ORDER_ITEM table correspond to the lines on such an order receipt.

The OrderNumber Column in ORDER_ITEM relates each row in ORDER_ITEM to the corresponding OrderNumber in the RETAIL_ORDER table. SKU identifies the actual item purchased by its stock-keeping unit number. Further, the SKU column in ORDER_ITEM relates each row in ORDER_ITEM to its corresponding SKU in the SKU_DATA table (discussed in the next section). Quantity is the number of items of that SKU purchased in that order. Price is the price of each item, and ExtendedPrice is equal to Quantity × Price.

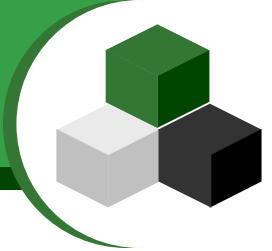
1. Cape Codd Outdoor Sports

➢ORDER_ITEM Data

ORDER_ITEM data are shown in the bottom part of Figure 2-4. The first row relates to order 1000 and to SKU 201000. For SKU 201000, one item was purchased for $300.00, and the ExtendedPrice was $300.00. The second row shows the second item in order 1000. There, 1 of item 202000 was purchased for $130.00 and the ExtendedPrice is 1 × $130.00, or $130.00. This table structure of an ORDER table related to an ORDER_ITEM table is typical for sales system with many items in one order. We will discuss it in detail in Chapters 5 and 6, where we will create a data model of a complete order and then design the database for that data model.

By the way You would expect the total of ExtendedPrice for all rows for a given order to equal OrderTotal in the RETAIL_ORDER table. They do not. For order 1000, for example, the sum of ExtendedPrice in the relevant rows of ORDER_ITEM is $300.00 + $130.00 = $430.00. However, the OrderTotal for order 1000 is $445.00. The difference occurs because OrderTotal includes tax, shipping, and other charges that do not appear in the data extract.

1. Cape Codd Outdoor Sports



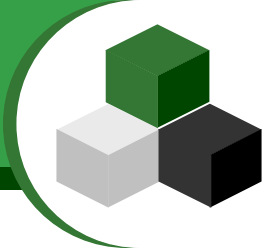
➢**SKU_DATA Table**

As shown in Figures 2-3 and 2-4, the SKU_DATA table has columns SKU, SKU_Description, Department, and Buyer:

SKU_DATA (SKU, SKU_Description, Department, Buyer)

SKU is an integer value that identifies a particular product sold by Cape Codd. For example, SKU 100100 identifies a yellow, standard-size SCUBA tank, whereas SKU 100200 identifies the magenta version of the same tank. SKU_Description contains a brief text description of each item. Department and Buyer identify the department and individual who is responsible for purchasing the product. As with the other tables, these columns are a subset of the SKU data stored in the operational database.

1. Cape Codd Outdoor Sports

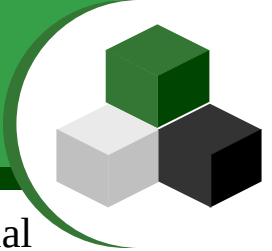


- **Data Extracts Are Common**

Before we continue, realize that the data extraction process described here is not just an academic exercise. To the contrary, such extraction processes are realistic, common, and important BI system operations. Right now, hundreds of businesses worldwide are using their BI systems to create extract databases just like the one created by Cape Codd.

In the next sections of this chapter, you will learn how to write SQL statements to process the extracted data via ad-hoc SQL queries, which is how SQL is used to “ask questions” about the data in the database. This knowledge is exceedingly valuable and practical. Again, right now, as you read this paragraph, hundreds of people are writing SQL to create information from extracted data. The SQL you will learn in this Chapter will be an essential asset to you as a knowledge worker, application programmer, or database administrator. Invest the time to learn SQL—the investment will pay great dividends later in your career.

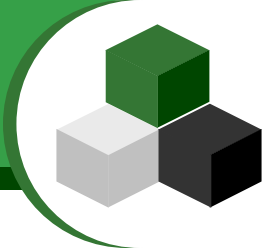
2. SQL Background



SQL was developed by the IBM Corporation in the late 1970s. It was endorsed as a national standard by the American National Standards Institute (ANSI) in 1986 and by the International Organization for Standardization (ISO) (and no, that's not a typo—the acronym is ISO, not IOS!) in 1987. Subsequent versions of SQL were adopted in 1989 and 1992. The 1992 version is sometimes referred to as SQL-92, or sometimes as ANSI-92 SQL. In 1999, SQL:1999 (also referred to as SQL3), which incorporated some object-oriented concepts, was released. This was followed by the release of SQL:2003 in 2003, SQL:2006 in 2006, and, most recently, SQL:2008 in 2008. Each of these added new features or extended existing SQL features, the most important of which for us is SQL support for Extensible Markup Language (XML). (XML is discussed in Chapter 12.) Our discussion in this chapter and in Chapter 7 focuses on common language features that have been in SQL since SQL-92, but does include some features from SQL:2003 and SQL:2008. We discuss the SQL XML features in Chapter12.

SQL is not a complete programming language, like Java or C#. Instead, it is called a data sublanguage, because it has only those statements needed for creating and processing database data and metadata. You can use SQL statements in many different ways. You can submit them directly to the DBMS for processing. You can embed SQL statements into client/server application programs. You can embed them into Web pages, and you can use them in reporting and data extraction programs. You also can execute SQL statements directly from Visual Studio.NET and other development tools.

2. SQL Background



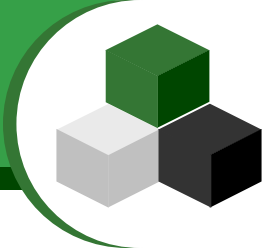
SQL statements are commonly divided into categories, three of which are of interest to us here:

- Data definition language (DDL) statements, which are used for creating tables, relationships, and other structures
- Data manipulation language (DML) statements, which are used for querying, inserting, modifying, and deleting data

This chapter considers only DML statements for querying data. The remaining DML statements for inserting, modifying, and deleting data are discussed in Chapter 7, where we will also discuss SQL DDL statements.

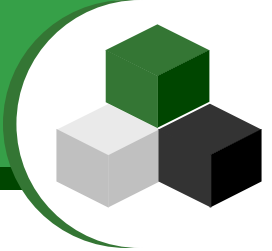
SQL is ubiquitous, and SQL programming is a critical skill. Today, nearly all DBMS products process SQL, with the only exceptions being some of the emerging NoSQL movement products. Enterprise-class DBMSs such as Microsoft SQL Server 2008 R2, Oracle Database 11g, Oracle MySQL 5.5, and IBM DB2 require that you know SQL. With these products, all data manipulation is expressed using SQL.

2. SQL Background



As explained in Chapter 1, if you have used Microsoft Access, you have used SQL, even if you didn't know it. Every time you process a form, create a report, or run a query Microsoft Access generates SQL and sends that SQL to Microsoft Access' internal ADE DBMS engine. To do more than elementary database processing, you need to uncover the SQL hidden by Microsoft Access. Further, once you know SQL, you will find it easier to write a query statement in SQL rather than fight with the graphical forms, buttons, and other paraphernalia that you must use to create queries with the Microsoft Access query-by-example style GUI.

3. The SQL SELECT/FROM/WHERE Framework

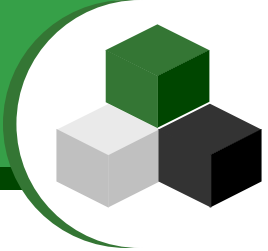


This section introduces the fundamental statement framework for SQL query statements. After we discuss this basic structure, you will learn how to submit SQL statements to Microsoft Access, SQL Server, Oracle Database, and MySQL. If you choose, you can then follow along with the text and process the SQL statements as they are explained in the rest of this chapter. The basic form of SQL queries uses the SQL SELECT/FROM/WHERE framework. In this framework:

- The SQL SELECT clause specifies which columns are to be listed in the query results.
- The SQL FROM clause specifies which tables are to be used in the query.
- The SQL WHERE clause specifies which rows are to be listed in the query results.

Let’s work through some examples so that this framework makes sense to you.

3. The SQL SELECT/FROM/WHERE Framework



- Reading Specified Columns from a Single Table

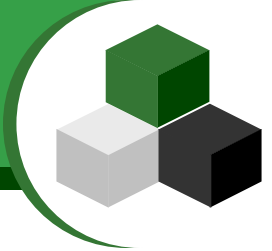
We begin very simply. Suppose we want to obtain just the values of the Department and Buyer columns of the SKU_DATA table. An SQL statement to read that data is the following:

```
SELECT    Department,  Buyer
  FROM    SKU_DATA;
```

Using the data in Figure 2-4, when the DBMS processes this statement the result will be:

	Department	Buyer
1	Water Sports	Pete Hansen
2	Water Sports	Pete Hansen
3	Water Sports	Nancy Meyers
4	Water Sports	Nancy Meyers
5	Camping	Cindy Lo
6	Camping	Cindy Lo
7	Climbing	Jerry Martin
8	Climbing	Jerry Martin

3. The SQL SELECT/FROM/WHERE Framework



- Reading Specified Columns from a Single Table

When SQL statements are executed, the statements transform tables. SQL statements start with a table, process that table in some way, and then place the results in another table structure. Even if the result of the processing is just a single number, that number is considered to be a table with one row and one column. As you will learn at the end of this chapter, some SQL statements process multiple tables. Regardless of the number of input tables, though, the result of every SQL statement is a single table.

Notice that SQL statements terminate with a semicolon (;) character. The semicolon is required by the SQL standard. Although some DBMS products will allow you to omit the semicolon, some will not, so develop the habit of terminating SQL statements with a semicolon.

3. The SQL SELECT/FROM/WHERE Framework

- Reading Specified Columns from a Single Table

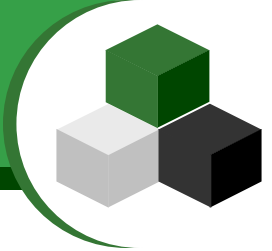
The order of the column names in the SELECT phrase determines the order of the columns in the results table. Thus, if we switch Buyer and Department in the SELECT phrase, they will be switched in the output table as well. Hence, the SQL statement:

```
SELECT    Buyer, Department
FROM  SKU_DATA;
```

produces the following result table:

	Buyer	Department
1	Pete Hansen	Water Sports
2	Pete Hansen	Water Sports
3	Nancy Meyers	Water Sports
4	Nancy Meyers	Water Sports
5	Cindy Lo	Camping
6	Cindy Lo	Camping
7	Jerry Martin	Climbing
8	Jerry Martin	Climbing

3. The SQL SELECT/FROM/WHERE Framework



- Reading Specified Columns from a Single Table

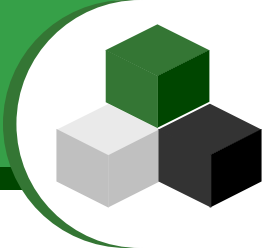
Notice that some rows are duplicated in these results. The data in the first and second row, for example, are identical. We can eliminate duplicates by using the SQL DISTINCT keyword, as follows:

```
SELECT    DISTINCT  Buyer,  Department
  FROM    SKU_DATA;
```

The result of this statement, where all of the duplicate rows have been removed, is:

	Buyer	Department
1	Cindy Lo	Camping
2	Jerry Martin	Climbing
3	Nancy Meyers	Water Sports
4	Pete Hansen	Water Sports

3. The SQL SELECT/FROM/WHERE Framework



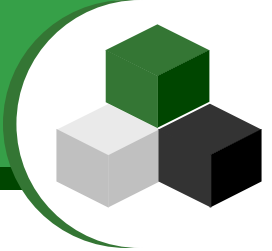
- Reading Specified Columns from a Single Table

By the way The reason that SQL does not automatically eliminate duplicate rows is that it can be very time consuming to do so. To determine if any rows are duplicates, every row must be compared with every other row. If there are 100,000 rows in a table, that checking will take a long time. Hence, by default duplicates are not removed. However, it is always possible to force their removal using the DISTINCT keyword.

Suppose that we want to view all of the columns of the SKU_DATA table. To do so, we can name each column in the SELECT statement as follows:

```
SELECT SKU, SKU_Description, Department, Buyer
FROM SKU_DATA;
```

3. The SQL SELECT/FROM/WHERE Framework



- Reading Specified Columns from a Single Table

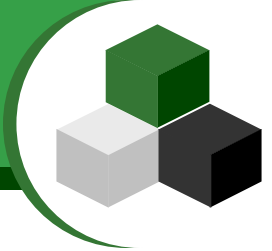
The result will be a table with all of the rows and all four of the columns in SKU_DATA:

	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
3	101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
4	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
5	201000	Half-dome Tent	Camping	Cindy Lo
6	202000	Half-dome Tent Vestibule	Camping	Cindy Lo
7	301000	Light Fly Climbing Harness	Climbing	Jerry Martin
8	302000	Locking Carabiner, Oval	Climbing	Jerry Martin

However, SQL provides a shorthand notation for querying all of the columns of a table. The shorthand is to use the SQL asterisk (*) wildcard character to indicate that we want all the columns to be displayed:

```
SELECT      *
  FROM      SKU_DATA;
```

3. The SQL SELECT/FROM/WHERE Framework

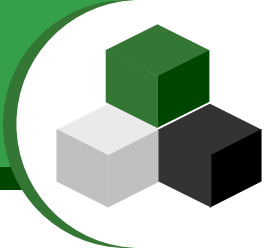


- Reading Specified Columns from a Single Table

  The result will be a table with all of the rows and all four of the columns in SKU_DATA:

	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
3	101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
4	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
5	201000	Half-dome Tent	Camping	Cindy Lo
6	202000	Half-dome Tent Vestibule	Camping	Cindy Lo
7	301000	Light Fly Climbing Harness	Climbing	Jerry Martin
8	302000	Locking Carabiner, Oval	Climbing	Jerry Martin

3. The SQL SELECT/FROM/WHERE Framework



- Reading Specified Rows from a Single Table

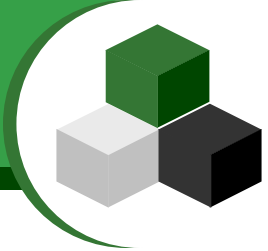
Suppose we want all of the columns of the SKU_DATA table, but we want only the rows for the Water Sports department. We can obtain that result by using the SQL WHERE clause as follows:

```
SELECT      *
FROM  SKU_DATA
WHERE       Department='Water  Sports';
```

The result of this statement will be:

	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
3	101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
4	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers

3. The SQL SELECT/FROM/WHERE Framework



- Reading Specified Rows from a Single Table

In an SQL WHERE clause, if the column contains text or date data, the comparison values must be enclosed in single quotation marks ('{text or date data}'). If the column contains numeric data, however, the comparison values need not be in quotes. Thus, to find all of the SKU rows with a value greater than 200,000, we would use the SQL statement (note that no comma is included in the numeric value code):

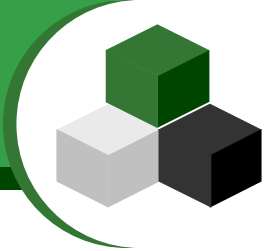
```
SELECT    *
  FROM    SKU_DATA
WHERE     SKU  >  200000;
```

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
3	301000	Light Fly Climbing Harness	Climbing	Jerry Martin
4	302000	Locking Carabiner, Oval	Climbing	Jerry Martin

By the way SQL is very fussy about single quotes. It wants the plain, nondirectional quotes found in basic text editors. The fancy directional quotes produced by many word processors will produce errors. For example, the data value 'Water Sports' is correctly stated, but ‘Water Sports’ is not. Do you see the difference?

3. The SQL SELECT/FROM/WHERE Framework



❖ Reading Specified Columns and Rows from a Single Table

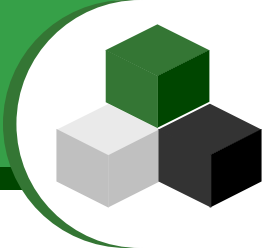
So far, we have selected certain columns and all rows and we have selected all columns and certain rows. We can combine these operations to select certain columns and certain rows by naming the columns we want and then using the SQL WHERE clause. For example, to obtain the SKU_Description and Department of all products in the Climbing department, we use the SQL query:

```
SELECT      SKU_Description,  Department
  FROM      SKU_DATA
WHERE       Department='Climbing';
```

The result is:

	SKU_Description	Department
1	Light Fly Climbing Harness	Climbing
2	Locking Carabiner, Oval	Climbing

3. The SQL SELECT/FROM/WHERE Framework



❖ Reading Specified Columns and Rows from a Single Table

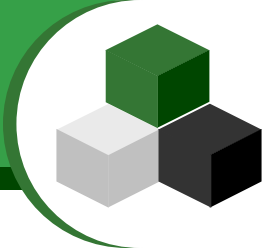
SQL does not require that the column used in the WHERE clause also appear in the SELECT clause column list. Thus, we can specify:

```
SELECT      SKU_Description,  Buyer
FROM  SKU_DATA
WHERE       Department='Climbing';
```

where the qualifying column, Department, does not appear in the SELECT clause column list. The result is:

	SKU_Description	Buyer
1	Light Fly Climbing Harness	Jerry Martin
2	Locking Carabiner, Oval	Jerry Martin

3. The SQL SELECT/FROM/WHERE Framework



- Reading Specified Columns and Rows from a Single Table

By the way Standard practice is to write SQL statements with the SELECT, FROM, and WHERE clauses on separate lines. This practice is just a coding convention, however, and SQL parsers do not require it. You could code SQL-Query-CH02-09 all on one line as:

```
SELECT SKU_Description, Buyer FROM SKU_DATA WHERE Department=
'Climbing';
```

All DBMS products would process the statement written in this fashion. However, the standard multiline coding convention makes SQL easier to read, and we encourage you to write your SQL according to it.

LOGO

Thank You!