HOUR 2

Defining Data Structures

What You'll Learn in This Hour:

- A look at the underlying data of a table
- An introduction to the basic data types
- Instruction on the use of various data types
- Examples depicting differences between data types

In this second hour, you learn more about the data you viewed at the end of Hour 1, "Welcome to the World of SQL." You learn the characteristics of the data and how such data is stored in a relational database. There are several data types, as you'll soon discover.

What Is Data?

Data is a collection of information stored in a database as one of several different data types. Data includes names, numbers, dollar amounts, text, graphics, decimals, figures, calculations, summarization, and just about anything else you can possibly imagine. Data can be stored in uppercase, lowercase, or mixed case. Data can be manipulated or changed; most data does not remain static for its lifetime.

Data types are used to provide rules for data for particular columns. A data type deals with the way values are stored in a column as far as the length allocated for a column and whether values such as alphanumeric, numeric, and date and time data are allowed. There is a data type for every possible bit or combination of data that can be stored in a particular database. These data types are used to store data such as characters, numbers, date and time, images, and other binary data. More specifically, the data might consist of names, descriptions, numbers, calculations, images, image descriptions, documents, and so forth.

The data is the purpose of any database and must be protected. The protector of the data is normally the *database administrator (DBA)*, although it is

every database user's responsibility to ensure that measures are taken to protect data. Data security is discussed in depth in Hour 18, "Managing Database Users," and Hour 19, "Managing Database Security."

Basic Data Types

The following sections discuss the basic data types supported by ANSI SQL. Data types are characteristics of the data itself, whose attributes are placed on fields within a table. For example, you can specify that a field must contain numeric values, disallowing the entering of alphanumeric strings. After all, you would not want to enter alphabetic characters in a field for a dollar amount. Defining each field in the database with a data type eliminates much of the incorrect data found in a database due to data entry errors. *Field definition* (data type definition) is a form of data validation that controls the type of data that may be entered into each given field.

Depending on your implementation of *relational database management system (RDBMS)*, certain data types can be converted automatically to other data types depending upon their format. This type of conversion in known as an *implicit conversion*, which means that the database handles the conversion for you. An example of this is taking a numeric value of 1000.92 from a numeric field and inputting it into a string field. Other data types cannot be converted implicitly by the host RDBMS and therefore must undergo an explicit conversion. This usually involves the use of an SQL function, such as `CAST` or `CONVERT`. For example

```
SELECT CAST('12/27/1974' AS DATETIME) AS MYDATE
```

The very basic data types, as with most other languages, are

- String types
- Numeric types
- Date and time types

Did You Know?

SQL Data Types

Every implementation of SQL seems to have its own specific set of data types. The use of implementation-specific data types is necessary to support the philosophy of each implementation on how to handle the storage of data. However, the basics are the same among all implementations.

Fixed-Length Strings

Constant characters, those strings that always have the same length, are stored using a fixed-length data type. The following is the standard for an SQL fixed-length character:

```
CHARACTER(n)
```

n represents a number identifying the allocated or maximum length of the particular field with this definition.

Some implementations of SQL use the `CHAR` data type to store fixed-length data. You can store alphanumeric data in this data type. An example of a constant length data type would be for a state abbreviation because all state abbreviations are two characters.

Spaces are normally used to fill extra spots when using a fixed-length data type; if a field's length was set to `10` and data entered filled only 5 places, the remaining 5 spaces would be recorded as spaces. The padding of spaces ensures that each value in a field is a fixed length.

Fixed-Length Data Types

Be careful not to use a fixed-length data type for fields that might contain varying-length values, such as an individual's name. If you use the fixed-length data type inappropriately, you eventually encounter problems such as the waste of available space and the inability to make accurate comparisons between data.

Always use the varying-length data type for nonconstant character strings to save database space.

Watch Out!

Varying-Length Strings

SQL supports the use of *varying-length strings*, strings whose length is not constant for all data. The following is the standard for an SQL varying-length character:

```
CHARACTER VARYING(n)
```

n represents a number identifying the allocated or maximum length of the particular field with this definition.

Common data types for variable-length character values are the `VARCHAR`, `VARBINARY`, and `VARCHAR2` data types. `VARCHAR` is the ANSI standard, which Microsoft SQL Server and MySQL use; Oracle uses both `VARCHAR` and `VARCHAR2`. The data stored in a character-defined column can be alphanumeric, which means that the data value may contain numeric characters. `VARBINARY` is similar to `VARCHAR` and `VARCHAR2` except that it contains a

variable length of bytes. Normally, you would use a type such as this to store some kind of digital data such as possibly an image file.

Remember that fixed-length data types typically pad spaces to fill in allocated places not used by the field. The varying-length data type does not work this way. For instance, if the allocated length of a varying-length field is `10`, and a string of 5 characters is entered, the total length of that particular value would be only `5`. Spaces are not used to fill unused places in a column.

Large Object Types

Some variable-length data types need to hold longer lengths of data than what is traditionally reserved for a `VARCHAR` field. The `BLOB` and `TEXT` data types are two examples of such data types in modern database implementations. These data types are specifically made to hold large sets of data. The `BLOB` is a binary large object, so its data is treated as a large binary string (a byte string). A `BLOB` is especially useful in an implementation that needs to store binary media files in the database, such as images or MP3s.

The `TEXT` data type is a large character string data type that can be treated as a large `VARCHAR` field. It is often used when an implementation needs to store large sets of character data in the database. An example of this would be storing HTML input from the entries of a blog site. Storing this type of data in the database enables the site to be dynamically updated.

Numeric Types

Numeric values are stored in fields that are defined as some type of number, typically referred to as `NUMBER`, `INTEGER`, `REAL`, `DECIMAL`, and so on.

The following are the standards for SQL numeric values:

- `BIT(`*n*`)`
- `BIT VARYING(`*n*`)`
- `DECIMAL(`*p*`,`*s*`)`
- `INTEGER`
- `SMALLINT`
- `BIGINT`
- `FLOAT(p,`*s*`)`
- `DOUBLE PRECISION(`*p*`,`*s*`)`
- `REAL(`*s*`)`

p represents a number identifying the allocated or maximum length of the particular field for each appropriate definition.

s is a number to the right of the decimal point, such as 34.*ss*.

A common numeric data type in SQL implementations is `NUMERIC`, which accommodates the direction for numeric values provided by ANSI. Numeric values can be stored as zero, positive, negative, fixed, and floating-point numbers. The following is an example using `NUMERIC`:

```
NUMERIC(5)
```

This example restricts the maximum value entered in a particular field to 99999. Note that all the database implementations that we use for the examples support the `NUMERIC` type but implement it as a `DECIMAL`.

Decimal Types

Decimal values are numeric values that include the use of a decimal point. The standard for a decimal in SQL follows, where *p* is the precision and *s* is the decimal's scale:

```
DECIMAL(p,s)
```

The *precision* is the total length of the numeric value. In a numeric defined `DECIMAL(4,2)`, the precision is 4, which is the total length allocated for a numeric value. The *scale* is the number of digits to the right of the decimal point. The scale is 2 in the previous `DECIMAL(4,2)` example. If a value has more places to the right side of the decimal point than the scale allows, the value is rounded; for instance, `34.33` inserted into a `DECIMAL(3,1)` is typically rounded to `34.3`.

If a numeric value was defined as the following data type, the maximum value allowed would be `99.99`:

```
DECIMAL(4,2)
```

The precision is 4, which represents the total length allocated for an associated value. The scale is 2, which represents the number of *places*, or *bytes*, reserved to the right side of the decimal point. The decimal point does not count as a character.

Allowed values for a column defined as `DECIMAL(4,2)` include the following:

- `12`
- `12.4`

- 12.44
- 12.449

The last numeric value, 12.449, is rounded off to 12.45 upon input into the column. In this case, any numbers between 12.445 and 12.449 would be rounded to 12.45.

Integers

An *integer* is a numeric value that does not contain a decimal, only whole numbers (both positive and negative).

Valid integers include the following:

- 1
- 0
- -1
- 99
- -99
- 199

Floating-Point Decimals

Floating-point decimals are decimal values whose precision and scale are variable lengths and virtually without limit. Any precision and scale is acceptable. The REAL data type designates a column with single-precision, floating-point numbers. The DOUBLE PRECISION data type designates a column that contains double-precision, floating-point numbers. To be considered a single-precision floating point, the precision must be between 1 and 21 inclusive. To be considered a double-precision floating point, the precision must be between 22 and 53 inclusive. The following are examples of the FLOAT data type:

- FLOAT
- FLOAT(15)
- FLOAT(50)

Date and Time Types

Date and time data types are quite obviously used to keep track of information concerning dates and time. Standard SQL supports what are called DATETIME data types, which include the following specific data types:

- DATE
- TIME
- DATETIME
- TIMESTAMP

The elements of a DATETIME data type consist of the following:

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND

By the Way

Date and Time Types

The SECOND element can also be broken down to fractions of a second. The range is from 00.000 to 61.999, although some implementations of SQL might not support this range. The extra 1.999 seconds is used for leap seconds.

Be aware that each implementation of SQL might have its own customized data type for dates and times. The previous data types and elements are standards to which each SQL vendor should adhere, but be advised that most implementations have their own data type for date values, varying in both appearance and the way date information is actually stored internally.

A length is not normally specified for a date data type. Later in this hour, you learn more about dates, how date information is stored in some implementations, and how to manipulate dates and times using conversion functions. You also study practical examples of how dates and times are used in the real world.

Literal Strings

A *literal string* is a series of characters, such as a name or a phone number, that is explicitly specified by a user or program. Literal strings consist of data with the same attributes as the previously discussed data types, but the value of the string is known. The value of a column is usually unknown because a column typically has a different value associated with each row of data in a table.

You do not actually specify data types with literal strings—you simply specify the string. Some examples of literal strings follow:

- `'Hello'`
- `45000`
- `"45000"`
- `3.14`
- `'November 1, 1997'`

The alphanumeric strings are enclosed by single quotation marks, whereas the number value `45000` is not. Also notice that the second numeric value of `45000` is enclosed by quotation marks. Generally speaking, character strings require quotation marks, whereas numeric strings don't.

The process that converts a number into a numeric type is known as an *implicit conversion*. This means that the database attempts to figure out what type it needs to create for the object. So if you do not have a number enclosed with single quotation marks, the SQL compiler assumes that you want a numeric type. You need to be careful when working with data to ensure that the data is being represented as you want it to be. Otherwise, it might skew your results or result in an unexpected error. You see later how literal strings are used with database queries.

NULL Data Types

As you should know from Hour 1, a `NULL` value is a missing value or a column in a row of data that has not been assigned a value. `NULL` values are used in nearly all parts of SQL, including the creation of tables, search conditions for queries, and even in literal strings.

The following are two methods for referencing a `NULL` value:

- `NULL` (the keyword `NULL` itself)

The following does not represent a NULL value, but a literal string containing the characters N-U-L-L:

```
'NULL'
```

When using the NULL data type, it is important to realize that data is not required in a particular field. If data is always required for a given field, always use NOT NULL with a data type. If there is a chance that there might not always be data for a field, it is better to use NULL.

BOOLEAN Values

A BOOLEAN value is a value of TRUE, FALSE, or NULL. BOOLEAN values are used to make data comparisons. For example, when criteria are specified for a query, each condition evaluates to a TRUE, FALSE, or NULL. If the BOOLEAN value of TRUE is returned by all conditions in a query, data is returned. If a BOOLEAN value of FALSE or NULL is returned, data might not be returned.

Consider the following example:

```
WHERE NAME = 'SMITH'
```

This line might be a condition found in a query. The condition is evaluated for every row of data in the table that is being queried. If the value of NAME is SMITH for a row of data in the table, the condition returns the value TRUE, thereby returning the data associated with that record.

Most database implementations do not implement a strict BOOLEAN type and instead opt to use their own methodology. MySQL contains the BOOLEAN type but it is merely a synonym for their existing TINYINT type. Oracle prefers to direct its users to use a CHAR(1) value to denote a BOOLEAN, and Microsoft SQL Server uses a value known as BIT.

By the Way

Differences in Data Type Implementations

Some of the data types mentioned during this hour might not be available by name in the implementation of SQL that you are using. Data types are often named differently among implementations of SQL, but the concept behind each data type remains. Most, if not all, data types are supported by relational databases.

User-Defined Types

A *user-defined type* is a data type that the user defines. User-defined types allow users to customize their own data types to meet data storage needs and are based on existing data types. User-defined data types can assist the

developer by providing greater flexibility during database application development because they maximize the number of possibilities for data storage. The `CREATE TYPE` statement is used to create a user-defined type.

For example, you can create a type as follows in both MySQL and Oracle:

```
CREATE TYPE PERSON AS OBJECT
(NAME       VARCHAR (30),
 SSN        VARCHAR (9));
```

You can reference your user-defined type as follows:

```
CREATE TABLE EMP_PAY
(EMPLOYEE   PERSON,
 SALARY     DECIMAL(10,2),
 HIRE_DATE  DATE);
```

Notice that the data type referenced for the first column `EMPLOYEE` is `PERSON`. `PERSON` is the user-defined type you created in the first example.

Domains

A *domain* is a set of valid data types that can be used. A domain is associated with a data type, so only certain data is accepted. After you create a domain, you can add constraints to the domain. Constraints work in conjunction with data types, allowing you to further specify acceptable data for a field. The domain is used like the user-defined type.

You can create a domain as follows:

```
CREATE DOMAIN MONEY_D AS NUMBER(8,2);
```

You can add constraints to your domain as follows:

```
ALTER DOMAIN MONEY_D
ADD CONSTRAINT MONEY_CON1
CHECK (VALUE > 5);
```

You can reference the domain as follows:

```
CREATE TABLE EMP_PAY
(EMP_ID        NUMBER(9),
EMP_NAME      VARCHAR2(30),
 PAY_RATE      MONEY_D);
```

Summary

Several data types are available with SQL. If you have programmed in other languages, you probably recognize many of the data types mentioned. Data types allow different types of data to be stored in the database, ranging from simple characters to decimal points to date and time. The concept of data types is the same in all languages, whether programming in a third-generation language such as C and passing variables or using a relational database implementation and coding in SQL. Of course, each implementation has its own names for standard data types, but they basically work the same. Also remember that an RDBMS does not have to implement all of the data types in the ANSI standard to be considered ANSI compliant. Therefore, it is prudent to check with the documentation of your specific RDBMS implementation to see what options you have available.

You must take care in planning for both the near and distant future when deciding on data types, lengths, scales, and precisions in which to store your data. Business rules and how you want the end user to access the data are other factors in deciding on specific data types. You should know the nature of the data and how data in the database is related to assign proper data types.

Q&A

Q. ***How is it that I can enter numbers such as a person's Social Security number in fields defined as character fields?***

A. Numeric values are still alphanumeric, which are allowed in string data types. The process is called an implicit conversion because the database system handles it automatically. Typically, the only data stored as numeric values are values used in computations. However, it might be helpful for some to define all numeric fields with a numeric data type to help control the data entered in that field.

Q. ***I still do not understand the difference between constant-length and varying-length data types. Can you explain?***

A. Say you have an individual's last name defined as a constant-length data type with a length of 20 bytes. Suppose the individual's name is Smith. When the data is inserted into the table, 20 bytes are taken: 5 for the name, and 15 for the extra spaces. (Remember that this is a constant-length data type.) If you use a varying-length data type with a length of 20 and insert `Smith`, only 5 bytes of space are taken. If you then imagine that you are inserting 100,000 rows of data into this system, you could possibly save 1.5 million bytes of data.

Q. *Are there limits on the lengths of data types?*

A. Yes, there are limits on the lengths of data types, and they do vary among the various implementations.

Workshop

The following workshop is composed of a series of quiz questions and practical exercises. The quiz questions are designed to test your overall understanding of the current material. The practical exercises are intended to afford you the opportunity to apply the concepts discussed during the current hour, as well as build upon the knowledge acquired in previous hours of study. Please take time to complete the quiz questions and exercises before continuing. Refer to Appendix C, "Answers to Quizzes and Exercises," for answers.

Quiz

1. True or false: An individual's Social Security number, entered in the format `'111111111'`, can be any of the following data types: constant-length character, varying-length character, or numeric.
2. True or false: The scale of a numeric value is the total length allowed for values.
3. Do all implementations use the same data types?
4. What are the precision and scale of the following?

```
DECIMAL(4,2)
DECIMAL(10,2)
DECIMAL(14,1)
```

5. Which numbers could be inserted into a column whose data type is `DECIMAL(4,1)`?

   A. `16.2`

   B. `116.2`

   C. `16.21`

   D. `1116.2`

   E. `1116.21`

6. What is data?

Exercises

1. Take the following column titles, assign them to a data type, decide on the proper length, and give an example of the data you would enter into that column.

   A. `ssn`

   B. `state`

   C. `city`

   D. `phone_number`

   E. `zip`

   F. `last_name`

   G. `first_name`

   H. `middle_name`

   I. `salary`

   J. `hourly_pay_rate`

   K. `date_hired`

2. Take the same column titles and decide whether they should be `NULL` or `NOT NULL`, realizing that in some cases where a column would normally be `NOT NULL`, the column could be `NULL` or vice versa, depending on the application.

   A. `ssn`

   B. `state`

   C. `city`

   D. `phone_number`

   E. `zip`

   F. `last_name`

   G. `first_name`

   H. `middle_name`

   I. `salary`

   J. `hourly_pay_rate`

   K. `date_hired`