Lab 1.1. Introduction to SQL

Ideally, a database language allows you to create database and table structures, perform basic data management chores (add, delete, and modify), and perform complex queries designed to transform the raw data into useful information. Moreover, a database language must perform such basic functions with minimal user effort, and its command structure and syntax must be easy to learn. Finally, it must be portable; that is, it must conform to some basic standard, so a person does not have to relearn the basics when moving from one RDBMS to another. SQL meets those ideal database language requirements well.

SQL functions fit into several broad categories:

Data manipulation language (DML). SQL includes commands to insert, update, delete, and retrieve data within the database tables.

Data definition language (DDL). SQL includes commands to create database objects such as tables, indexes, and views, as well as commands to define access rights to those database objects.

Transaction control language (TCL). The DML commands in SQL are executed within the context of a transaction, which is a logical unit of work composed of one or more SQL statements, as defined by business rules. SQL provides commands to control the processing of these statements an indivisible unit of work.

Data control language (DCL). Data control commands are used to control access to data objects, such as giving a one user permission to only view the PRODUCT table, and giving another user permission to change the data in the PRODUCT table.

Data Types

A data type is a specification about the kinds of data that can be stored in an attribute. Data types influence queries that retrieve data because there are slight differences in the syntax of SQL and how it behaves during a query that are based on the data type of the column being retrieved.

Three fundamental types of data:

character data

numeric data

date data

Character data is composed of any printable characters such as alphabetic values, digits, punctuation, and special characters. Character data is also often referred to as a "string" because it is a collection of characters threaded together to create the value.

Numeric data is composed of digits, such that the data has a specific numeric value.

Date data is composed of date and, occasionally, time values.

Note: Although character data may contain digits, the DBMS does not recognize the numeric value of those digits.

SOME COMMON SQL DATA TYPES

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D) or NUMERIC(L,D)	The declaration NUMBER(7,2) or NUMERIC(7,2) indicates that numbers will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place (for example, 12.32 or –134.99).
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as *Smith* and *Katzenjammer* are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) or VARCHAR(25) will let you store characters up to 25 characters long. However, unlike CHAR, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.
Date	DATE	Stores dates in the Julian date format.

Lab 1.2. SQL Data Definition Commands

SQL DATA DEFINITION COMMANDS

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	**Creates a database schema**
CREATE TABLE	**Creates a new table in the user's database schema**
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	**Creates an index for a table**
CREATE VIEW	**Creates a dynamic subset of rows and columns from one or more tables**
ALTER TABLE	**Modifies a table's definition (adds, modifies, or deletes attributes or constraints)**
CREATE TABLE AS	**Creates a new table based on a query in the user's database schema**
DROP TABLE	**Permanently deletes a table (and its data)**
DROP INDEX	**Permanently deletes an index**
DROP VIEW	**Permanently deletes a view**

Lab 1.3. SQL Data Manipulation Commands

Lab 1.4. Starting Database Model

Lab 1.5. Data Dictionary

In the data dictionary, note the data types. Keep in mind that data-type selection is usually dictated by the nature and intended use of the data. For example:

• P_PRICE clearly requires some kind of numeric data type; defining it as a character field is not acceptable.

• Just as clearly, a vendor name is an obvious candidate for a character data type. For example, VARCHAR(35) fits well because vendor names are variable-length character strings, and in this case, such strings may be up to 35 characters long.

• At first glance, it might seem logical to select a numeric data type for V_AREACODE because it contains only digits. However, adding and subtracting area codes does not yield meaningful results. Therefore, selecting a character data type is more appropriate. This is true for many common attributes found in business data models. For example, even though zip codes contain all digits, they must be defined as character data because some zip codes begin with the digit zero (0), and a numeric data type would cause the leading zero to be dropped.

• U.S. state abbreviations are always two characters, so CHAR(2) is a logical choice.

• Selecting P_INDATE to be a (Julian) DATE field rather than a character field is desirable because Julian dates allow you to make simple date comparisons and perform date arithmetic. For instance, if you have used DATE fields, you can determine the number of days between dates.

TABLE 8.2

DATA DICTIONARY FOR THE CH08_SALECO DATABASE

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE	REQUIRED	PK OR FK	FK REFERENCED TABLE
PRODUCT	P_CODE	Product code	VARCHAR(10)	XXXXXXXXXX	NA	Y	PK	
	P_DESCRIPT	Product description	VARCHAR(35)	Xxxxxxxxxxx	NA	Y		
	P_INDATE	Stocking date	DATE	DD-MON-YYYY	NA	Y		
	P_QOH	Units available	SMALLINT	####	0–9999	Y		
	P_MIN	Minimum units	SMALLINT	####	0–9999	Y		
	P_PRICE	Product price	NUMBER(8,2)	####.##	0.00–9999.00	Y		
	P_DISCOUNT	Discount rate	NUMBER(5,2)	0.##	0.00–0.20	Y		
	V_CODE	Vendor code	INTEGER	###	100–999		FK	VENDOR
VENDOR	V_CODE	Vendor code	INTEGER	#####	1000–9999	Y	PK	
	V_NAME	Vendor name	VARCHAR(35)	Xxxxxxxxxxxxx	NA	Y		
	V_CONTACT	Contact person	VARCHAR(25)	Xxxxxxxxxxxxx	NA	Y		
	V_AREACODE	Area code	CHAR(3)	999	NA	Y		
	V_PHONE	Phone number	CHAR(8)	999–9999	NA	Y		
	V_STATE	State	CHAR(2)	XX	NA	Y		
	V_ORDER	Previous order	CHAR(1)	X	Y or N	Y		

FK = Foreign key
PK = Primary key
CHAR = Fixed-length character data, 1 to 255 characters
VARCHAR = Variable-length character data, 1 to 2,000 characters. VARCHAR is automatically converted to VARCHAR2 in Oracle.
NUMBER = Numeric data. NUMBER(9,2) is used to specify numbers that have two decimal places and are up to nine digits long, including the decimal places. Some RDBMSs permit the use of a MONEY or a CURRENCY data type.
NUMERIC = Numeric data. DBMSs that do not support the NUMBER data type typically use NUMERIC instead.
INT = Integer values only. INT is automatically converted to NUMBER in Oracle.
SMALLINT = Small integer values only. SMALLINT is automatically converted to NUMBER in Oracle.
DATE formats vary. Commonly accepted formats are DD-MON-YYYY, DD-MON-YY, MM/DD/YYYY, and MM/DD/YY.

*Not all the ranges shown here will be illustrated in this chapter. However, you can use these constraints to practice writing your own.

Data-type selection sometimes requires professional judgment. For example, you must make a decision about the V_CODE's data type as follows:

• If you want the computer to generate new vendor codes by adding 1 to the largest recorded vendor code, you must classify V_CODE as a numeric attribute. (You cannot perform mathematical procedures on character data.) The designation INTEGER will ensure that only the counting numbers (integers) can be used. Most SQL implementations also permit the use of SMALLINT for integer values up to six digits.

• If you do not want to perform mathematical procedures based on V_CODE, you should classify it as a character attribute, even though it is composed entirely of numbers. When there is no need to perform mathematical procedures on the attribute, store it as a character attribute.

When you define the attribute's data type, you must pay close attention to the expected use of the attributes for sorting and data-retrieval purposes. For example, in a real estate application, an attribute that represents the numbers of bathrooms in a home (H_BATH_NUM) could be assigned the CHAR(3)

data type because the application will probably not do any addition, multiplication, or division with the number of bathrooms. Based on the CHAR(3) data-type definition, valid H_BATH_NUM values would be '2','1','2.5','10'. However, this data-type decision creates potential problems. For example, if an application sorts the homes by number of bathrooms, a query would "see" the value '10' as less than '2', which is clearly incorrect. So, you must consider the expected use of the data to properly define the attribute data type.

Source:

Coronel, C., & Morris, S. (2019). Database systems: design, implementation, and management. Boston: Cengage.

Lab 1.6. Creating the Database

Before you can use a new RDBMS, you must complete two tasks: create the database structure and create the tables that will hold the end-user data. To complete the first task, the RDBMS creates the physical files that will hold the database. When you create a new database, the RDBMS automatically creates the data dictionary tables in which to store the metadata and creates a default database administrator. Creating the physical files that will hold the database means interacting with the operating system and the file systems supported by the operating system. Therefore, creating the database structure is one feature that tends to differ substantially from one RDBMS to another. However, it is relatively easy to create a database structure, regardless of which RDBMS you use.

If you use Microsoft Access, creating the database is simple: start Access and open a new blank database. Specify the folder in which you want to store the database, and then name the database.

However, if you work in a database environment typically used by larger organizations, you will probably use an enterprise RDBMS such as Oracle, MS SQL Server, MySQL, or DB2. Given their security requirements and greater complexity, creating a database with these products is a more elaborate process.

With the exception of creating the database, most RDBMS vendors use SQL that deviates little from the ANSI standard SQL. For example, most RDBMSs require each SQL command to end with a semicolon. However, some SQL implementations do not use a semicolon.

Source:

Coronel, C., & Morris, S. (2019). Database systems: design, implementation, and management. Boston: Cengage.

Lab 1.7. Creating Table Structures

Creating Table Structures

Once the database has been created and the appropriate data types for each attribute have been determined, it is time to create the actual database tables. Recall that when implementing the database design, every entity becomes a table, and the attributes of each entity become the columns in that table.

CREATE TABLE command

Now you are ready to implement the PRODUCT and VENDOR table structures with the help of SQL, using the CREATE TABLE syntax shown next.

```
CREATE TABLE tablename (
column1         data type       [constraint] [,
column2         data type       [constraint] ] [,
PRIMARY KEY     (column1        [, column2]) ] [,
FOREIGN KEY     (column1        [, column2]) REFERENCES tablename] [,
CONSTRAINT      constraint ] );
```

To make the SQL code more readable, most SQL programmers use one line per column (attribute) definition. In addition, spaces are used to line up the attribute characteristics and constraints. Finally, both table and attribute names are fully capitalized. Those conventions are used in the following examples that create VENDOR and PRODUCT tables and subsequent tables.

```
CREATE TABLE VENDOR (
V_CODE          INTEGER         NOT NULL      UNIQUE,
V_NAME          VARCHAR(35)     NOT NULL,
V_CONTACT       VARCHAR(25)     NOT NULL,
V_AREACODE      CHAR(3)         NOT NULL,
V_PHONE         CHAR(8)         NOT NULL,
V_STATE         CHAR(2)         NOT NULL,
V_ORDER         CHAR(1)         NOT NULL,
PRIMARY KEY (V_CODE));

CREATE TABLE PRODUCT (
P_CODE          VARCHAR(10)     NOT NULL      UNIQUE,
P_DESCRIPT      VARCHAR(35)     NOT NULL,
P_INDATE        DATE            NOT NULL,
P_QOH           SMALLINT        NOT NULL,
P_MIN           SMALLINT        NOT NULL,
P_PRICE         NUMBER(8,2)     NOT NULL,
P_DISCOUNT      NUMBER(5,2)     NOT NULL,
V_CODE          INTEGER,
PRIMARY KEY (P_CODE),
FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE);
```

As you examine the preceding SQL table-creating command sequences, note the following features:

• The NOT NULL specifications for the attributes ensure that a data entry will be made. When it is crucial to have the data available, the NOT NULL specification will not allow the end user to leave the attribute empty (with no data entry at all). Because this specification is made at the table level and stored in the data dictionary, application programs can use this information to create the data dictionary validation automatically.

• The UNIQUE specification creates a unique index on the respective attribute. Use it to avoid having duplicated values in a column.

• The primary key attributes contain both a NOT NULL and UNIQUE specification, which enforce the entity integrity requirements. If the NOT NULL and UNIQUE specifications are not supported, use

PRIMARY KEY without the specifications. (For example, if you designate the PK in MS Access, the NOT NULL and UNIQUE specifications are automatically assumed and are not spelled out.)

• The entire table definition is enclosed in parentheses. A comma is used to separate each table element definition (attributes, primary key, and foreign key).

•The ON UPDATE CASCADE specification ensures that if you make a change in any VENDOR's V_CODE that change is automatically applied to all foreign key references throughout the system to ensure that referential integrity is maintained. (Although the ON UPDATE CASCADE clause is part of the ANSI standard, some RDBMSs, such as Oracle, do not support it. If your RDBMS does not support the clause, delete it from the code shown here.)

• An RDBMS automatically enforces referential integrity for foreign keys. That is, you cannot have an invalid entry in the foreign key column; at the same time, you cannot delete a vendor row as long as a product row references that vendor.

• The command sequence ends with a semicolon. (Remember that your RDBMS may require you to omit the semicolon.)

Lab 1.8. Adding/Inserting Table Rows

SQL requires the use of the INSERT command to enter data into a table. The INSERT command's basic syntax looks like this:

INSERT INTO tablename VALUES (value1, value2, …, valuen)

Because the PRODUCT table uses its V_CODE to reference the VENDOR table's V_CODE, an integrity violation will occur if the VENDOR table V_CODE values do not yet exist. Therefore, you need to enter the VENDOR rows before the PRODUCT rows. Given the VENDOR table structure defined earlier and the sample VENDOR data shown in Figure, you would enter the first two data rows as follows:

INSERT INTO VENDOR

VALUES (21225',Bryson, Inc.','Smithson','615','223-3234','TN','Y');

INSERT INTO VENDOR

```
    VALUES (21226',Superloo, Inc.','Flushing','904','215-8995','FL','N');
```

and so on, until all of the VENDOR table records have been entered.

The PRODUCT table rows would be entered in the same fashion, using the PRODUCT data shown in Figure 8.2. For example, the first two data rows would be entered as follows, pressing Enter at the end of each line:

```
INSERT INTO PRODUCT
    VALUES ('11QER/31','Power painter, 15 psi., 3-nozzle','03-Nov-17',
    8,5,109.99,0.00,25595);
INSERT INTO PRODUCT
    VALUES ('13-Q2/P2','7.25-in. pwr. saw blade','13-Dec-17',32,15,14.99, 0.05,        21344);
```

Inserting Rows with Null Attributes

Thus far, you have entered rows in which all of the attribute values are specified. But what do you do if a product does not have a vendor or if you do not yet know the vendor code? In those cases, you would want to leave the vendor code null. To enter a null, use the following syntax:

```
INSERT INTO PRODUCT
    VALUES ('BRT-345','Titanium drill bit','18-Oct-17', 75, 10, 4.50, 0.06, NULL);
```

Incidentally, note that the NULL entry is accepted only because the V_CODE attribute is optional—the NOT NULL declaration was not used in the CREATE TABLE statement for this attribute.

Inserting Rows with Optional Attributes

Sometimes, more than one attribute is optional. Rather than declaring each attribute as NULL in the INSERT command, you can indicate just the attributes that have required values. You do that by listing the attribute names inside parentheses after the table name. For the purpose of this example, assume that the only required attributes for the PRODUCT table are P_CODE and P_DESCRIPT:

```
INSERT INTO PRODUCT(P_CODE, P_DESCRIPT) VALUES ('BRT-345','Titanium drill bit');
```