Database Management Systems

Structured Query Language (SQL): Data Management
Sotirios Zygiaris,

Chapter 3

Structured Query Language (SQL): Data Management

3.1. SQL Basics

SQL (Structured Query Language) or sequel is command language that enables the database users to create, manipulate data, and extract information. It uses an English-like command set and is easy to learn. The language provides direct access to managers to question the database and produce ad-hoc information reports.

SQL provides access to the database, but does not produce any applications with menus, windows, etc. It is the role of application developers to connect the database with the applications. Moreover, it is a powerful tool in the hands of managers, who are familiar with SQL, to manage the accessibility to their company data and produce on-demand managerial reports that support decision-making.

SQL has different uses for managing data, producing information, and administering the database. Although SQL is prescribed universally by the American National Standards Institute (ANSI) as SQL-2003, there many dialects used by the various RDBMS vendors. This book follows the standard SQL. Moreover, any variations from existing standards will be indicated across the three RDBS (MS-Access, Oracle, and MariaDB) used in this textbook. This chapter explores data management SQL topics for creating data structures and updating data.

SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. During the 1970s, ORACLE introduced their SQL system and IBM introduced the R version of SQL. Many software vendors followed, including MariaDB, as an open-source platform.

The creation of database in RDBMS involves three important steps, namely, the creation of the: **(1) database, (2) database tables, and (3) the creation of the relationships between these tables.**

3.2. Creating the Database

The SQL command: CREATE **`DATABASE Atzainc`**; will create a new database named AtzaInc. The keywords CREATE DATABASE are reserved words by SQL and the identifier Atzainc is selected by the database creator. SQL is not case sensitive, meaning that there is no difference in execution when the user uses small or capital letters. For example, the use of ATZAinc, AtzaINC

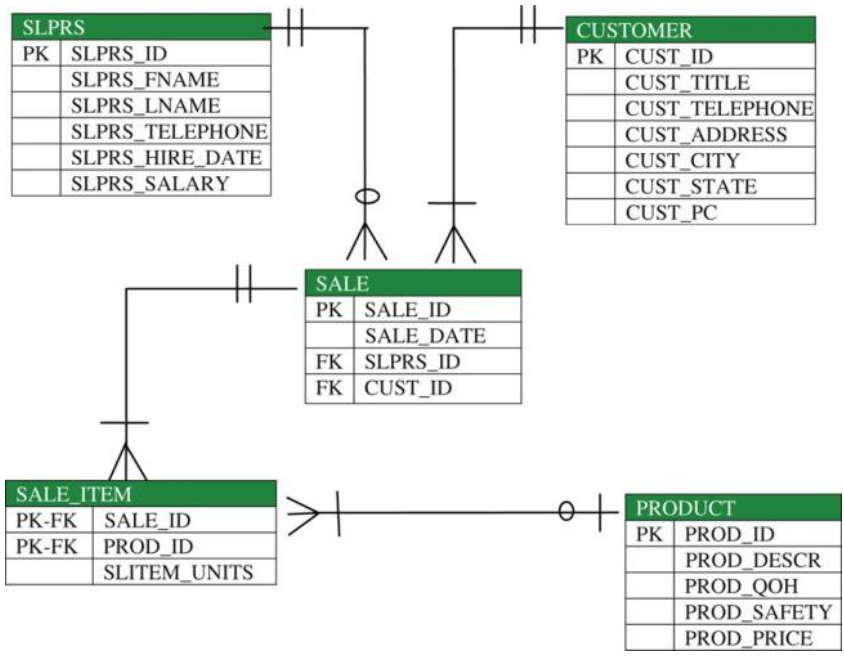


Figure 3.1. ERM Diagram for Atza Inc.

produces the same result. Users may use many empty spaces between keywords. The SQL commands always ends with a semicolon in standard SQL.

In MS-Access, a new database is created visually with simple steps. Just go to the file selection screen and select new. In my SQL, it is also a simple task. Just select new in the PHPMyAdmin window and enter the database name. ORACLE does not require a database creation as it accumulates all user tables under a common workspace. The installer program creates a database named XE, and inside that database is a schema named HRT. **(See Videos for MariaDB, ORACLE, and MS-Access for database creation.)**

Figure 3.1 is a reminder of the ERM designed for AtzaInc. This example will be used for practical exercises throughout this textbook.

3.2.1. Creating the Database Tables

Once the database is created, the next task is to create the structure of the tables using the CREATE TABLE SQL command. The table creation follows the ERM for the Atzainc, Figure 3.1, as a blueprint, the same way engineers are using an architectural blueprint to build a house. The following SQL command is used to create a table in standard SQL. Capital letters designate the SQL reserved words; lower case and italicized letters are the user-defined identifier names selected by the database designer.

```
CREATE TABLE Tablename
{
Columnname1 TYPE CONSTRAINT(s),
ColumnnameX TYPE CONSTRAINT(s),
PRIMARY KEY (Columnname1, ColumnnameX), CONSTRAINT(s),
FOREIGN KEY (Columnname) REFERENCES Tablename,
CONSTRAINT CONSTRAINT1..CONSTRAINTX
};
```

CREATE TABLE Tablename, the user defines the table, which must be descriptive of the data in the table stores. Abbreviations may be used like SLPRS for salesperson. It is a good practice to remain consistent in the table naming with capital or small letters. The SQL command begins with {character and ends with} followed by a semicolon.

Columnname is the name of the table's attributes that is to be given in the E−R diagram. Again, employing descriptive names that reveal the data that the attribute holds are imperative. **TYPE** is the type of data that the attribute stores. These data types could be as follows (see Table 3.1).

CONSTRAINT(s), Constrains are limitations that the database designers define on the data sets. The constraints could be as follows (see Table 3.2):

A detailed description of SQL constraints follows along with the different versions that these constraints apply in each of the three DBMS examined in this chapter. The order of creation of tables must follow a certain order. First, the tables having a primary key that is used as a foreign key must be created.

Table 3.1. Most Used SQL Data Types.

Data Type	Format	Description
NUMERIC	NUMBER (length, decimals)	The declaration NUMBER (5,1) a decimal number that occupies five spaces including the sign and the decimal place. The decimal part uses one space
		Examples 1427.2, -234.7, 78.7
	INTEGER or INT	Used to store whole numbers like 7854−5529
TEXT	CHAR (length)	CHAR (10) can store up to 10 characters. If only four characters are used in the data field, the rest of the six spaces will remain unused
	VARCHAR (length) or VARCHAR2 (length)	VARCHAR (!0) will store up to 10 characters but not leaving unused spaces. VARCHAR2 stores up to 25 characters
DATE	DATE	Stores formatted dates

Table 3.2. SQL Constraints.

NOT NULL	Does not allow a column to have empty data. For example, in the Salesperson table, first name and last name are not allowed to be left empty of data. For numeric types, 0 is considered to be a value. For text types, an empty space " " is also a value. **Primary keys are by default NOT NULL**
UNIQUE	Like primary keys data, unique is not allowed duplicate values. For example, in salesperson person, if the telephone constraint is set to unique, it is not allowed for two salespersons to have the same telephone number
DEFAULT	A default value is placed when no data are entered. For example, salary could have a default value of 10
CHECK	Validated data, for example, hire date in the salesperson file, and could not be after today's date
INDEX	Creates an index of a table in a specific column
ON DELETE, ON UPDATE	**On delete Cascade** deletes foreign keys when a primary key is deleted
	On update Cascade updates foreign keys when a primary key is updated
	On delete/update Restrict does not allow deletion or updating of foreign keys when a primary key is deleted or updated
	On delete/update Set NULL sets to NULL foreign keys when a primary key is deleted or updated
PRIMARY KEY	It defines the column or the columns in the case of a composite primary key that uniquely identifies each column. By default, the primary key is unique and NOT NULL. Only one primary key constraint is allowed for each table
FOREIGN KEY	It defines the foreign key(s) that exist on a table. For every foreign key, a different constraint must be created referencing the related table that the foreign key exists as a primary key

For example, the primary key of SALESPERSON is a foreign key in the Sale table. Therefore, the SALESPERSON table is created first using the CREATE TABLE SQL command as it is described below.

Every table must be specified with a primary key constraint that uniquely identifies each row or tulle of the data and must be specified inside the CREATE TABLE command as:

In MariaDB
PRIMARY KEY (SLPRS_ID)

In ORACLE
CONSTRAINT S1 PRIMARY KEY (SLPRS_ID)

In MS-Access
SLPRS_ID INTEGER CONSTRAINT S1 PRIMARY KEY or
CONSTRAINT S1 PRIMARY KEY (SLPRS_ID)

When a composite primary key must be specified, just include the two keys separated by a comma. For example:

CONSTRAINT S1 PRIMARY KEY (SLPRS_ID, SLPRS_TELEPHONE)

Establishing the relationship between two tables requires the creation of the foreign key inside the table of the highest multiplicity using the foreign key constraint inside the CREATE TABLE command.

MariaDB
FOREIGN KEY (SLPRS_ID) REFERENCES SLPRS(SLPRS_ID)

ORACLE, MS-ACCESS
CONSTRAINT C1 FOREIGN KEY (SLPRS_ID) REFERENCES SLPRS (SLPRS_ID)

Some issues that must be noted in relation to the primary and foreign constraints are:

- Constraint C1 can be any legal identifier name that is unique for every constraint. **A table must have one and only one primary key but may have zero, one, or many foreign keys.**
- In foreign key constraints, the name of related columns **may not be the same.** For example, FOREIGN KEY (**SLPRS_ID**) REFERENCES SLPRS (**S_ID**) is legal, assuming the primary key in the SLPRS table is referred to as S_ID.
- Tables are created in a certain order. First, tables that contribute to a relationship with their primary keys must be created. Then tables that include foreign keys must be created. Otherwise, there will an error for reference to a nonexisting table. In the SalesInc example, first, the Salesperson, Customer, and Product tables are created and then the Sale and Sale Items table.

Table 3.3 presents the CREATE TABLE SQL command for the AtzaInc case study in MariaDB, ORACLE, and MS-Access.

3.2.1.1. Creating Tables in MariaDB

The creation of database tables in MariaDB could be implemented using two methods:

(1) Direct execution of the CREATE SQL command through the SQL command editor or
(2) Visual creation of database tables through the phpMyAdmin interactive environment.

Table 3.3. Creation of Atza Inc. Tables in MariaDB, Oracle, and MS-Access.

MariaDB	ORACLE	MS-Access
CREATE TABLE SLPRS (SLPRS_ID INT, SLPRS_FNAME VARCHAR(20) NOT NULL, SLPRS_LNAME VARCHAR(20) NOT NULL, SLPRS_TELEPHONE VARCHAR(12) UNIQUE, SLPRS_HIRE_DATE DATE, SLPRS_SALARY INT NOT NULL DEFAULT 600, PRIMARY KEY (SLPRS_ID))	CREATE TABLE SLPRS (SLPRS_ID INT, SLPRS_FNAME VARCHAR(20) NOT NULL, SLPRS_LNAME VARCHAR(20) NOT NULL, SLPRS_TELEPHONE VARCHAR(12) UNIQUE, SLPRS_HIRE_DATE DATE, SLPRS_SALARY INT NOT NULL, CONSTRAINT S1 PRIMARY KEY (SLPRS_ID))	CREATE TABLE SLPRS (SLPRS_ID INTEGER CONSTRAINT S1 PRIMARY KEY, SLPRS_FNAME CHAR(20) NOT NULL, SLPRS_LNAME CHAR(20) NOT NULL, SLPRS_TELEPHONE CHAR(12) UNIQUE, SLPRS_HIRE_DATE DATE, SLPRS_SALARY INTEGER NOT NULL)
CREATE TABLE CUSTOMER (CUST_ID INT NOT NULL, CUST_TITLE VARCHAR(30) NOT NULL, CUST_TELEPHONE VARCHAR(13) NOT NULL, CUST_ADDRESS VARCHAR(20) NOT NULL, CUST_CITY VARCHAR(20) NOT NULL, CUST_STATE VARCHAR(20) NULL, CUST_PC VARCHAR(5), PRIMARY KEY (CUST_ID))	CREATE TABLE CUSTOMER (CUST_ID INTEGER CONSTRAINT C1 PRIMARY KEY, CUST_TITLE VARCHAR(30) NOT NULL, CUST_TELEPHONE VARCHAR(13) NOT NULL, CUST_ADDRESS VARCHAR(20) NOT NULL, CUST_CITY VARCHAR(20) NOT NULL, CUST_STATE VARCHAR(20) NULL, CUST_PC VARCHAR(5))	CREATE TABLE CUSTOMER (CUST_ID INTEGER CONSTRAINT C1 PRIMARY KEY, CUST_TITLE CHAR(30) NOT NULL, CUST_TELEPHONE CHAR(13) NOT NULL, CUST_ADDRESS CHAR(20) NOT NULL, CUST_CITY CHAR(20) NOT NULL, CUST_STATE CHAR(20) NULL, CUST_PC CHAR(5))
CREATE TABLE PRODUCT (PROD_ID INT NOT NULL, PROD_DESCR VARCHAR(50) NOT NULL, PROD_QOH INT NOT NULL, PROD_SAFETY INT NOT NULL DEFAULT 10, PROD_PRICE INT, PRIMARY KEY (PROD_ID))	CREATE TABLE PRODUCT (PROD_ID INTEGER CONSTRAINT P1 PRIMARY KEY, PROD_DESCR VARCHAR(50) NOT NULL, PROD_QOH INTEGER NOT NULL, PROD_SAFETY INTEGER NOT NULL, PROD_PRICE INTEGER)	CREATE TABLE PRODUCT (PROD_ID INTEGER CONSTRAINT P1 PRIMARY KEY, PROD_DESCR CHAR(50) NOT NULL, PROD_QOH INTEGER NOT NULL, PROD_SAFETY INTEGER NOT NULL, PROD_PRICE INTEGER)
CREATE TABLE SALE (SALE_ID INT NOT NULL, SALE_DATE DATE NOT NULL, SLPRS_ID INT NOT NULL, CUST_ID INT NOT NULL, PRIMARY KEY (SALE_ID), INDEX SL1INDEX (SALE_DATE), FOREIGN key (SLPRS_ID) REFERENCES SLPRS(SLPRS_ID) ON DELETE RESTRICT ON UPDATE CASCADE, FOREIGN KEY (CUST_ID) REFERENCES CUSTOMER(CUST_ID) ON DELETE RESTRICT ON UPDATE CASCADE)	CREATE TABLE SALE (SALE_ID INTEGER CONSTRAINT SL1 PRIMARY KEY, SALE_DATE DATE NOT NULL, SLPRS_ID INTEGER NOT NULL, CUST_ID INTEGER NOT NULL, CONSTRAINT SL2 FOREIGN KEY (SLPRS_ID) REFERENCES SLPRS(SLPRS_ID), CONSTRAINT SL3 FOREIGN KEY (CUST_ID) REFERENCES CUSTOMER(CUST_ID) ON DELETE RESTRICT)	CREATE TABLE SALE (SALE_ID INTEGER CONSTRAINT SL1 PRIMARY KEY, SALE_DATE DATE NOT NULL, SLPRS_ID INTEGER NOT NULL, CUST_ID INTEGER NOT NULL, CONSTRAINT SL2 FOREIGN KEY (SLPRS_ID) REFERENCES SLPRS(SLPRS_ID), CONSTRAINT SL3 FOREIGN KEY (CUST_ID) REFERENCES CUSTOMER(CUST_ID))
CREATE TABLE SALE_ITEM (SALE_ID INT NOT NULL, PROD_ID INT NOT NULL, SLITEM_UNITS INT NOT NULL, PRIMARY KEY (SALE_ID, PROD_ID), FOREIGN KEY (SALE_ID) REFERENCES SALE(SALE_ID) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (PROD_ID) REFERENCES PRODUCT(PROD_ID) ON DELETE CASCADE ON UPDATE CASCADE)	CREATE TABLE SALE_ITEM (SALE_ID INTEGER, PROD_ID INTEGER, SLITEM_UNITS INTEGER NOT NULL, CONSTRAINT SLI1 PRIMARY KEY (SALE_ID, PROD_ID), CONSTRAINT SL5 FOREIGN KEY (SALE_ID) REFERENCES SALE(SALE_ID) ON DELETE CASCADE, CONSTRAINT SL6 FOREIGN KEY (PROD_ID) REFERENCES PRODUCT(PROD_ID) ON DELETE CASCADE)	CREATE TABLE SALE_ITEM (SALE_ID INTEGER, PROD_ID INTEGER, SLITEM_UNITS INTEGER NOT NULL, CONSTRAINT SLI1 PRIMARY KEY (SALE_ID, PROD_ID), CONSTRAINT SL5 FOREIGN KEY (SALE_ID) REFERENCES SALE(SALE_ID), CONSTRAINT SL6 FOREIGN KEY (PROD_ID) REFERENCES PRODUCT(PROD_ID))

Note: See file Createtables.txt

This chapter examines both methods; students may use the manual method to create a number of tables and the visual method to create other tables.

Creating Tables in MariaDB through the SQL Command Editor. Open PHPMyAdmin and insert in the SQL window (see Figure 3.2) the SQL command for table creation. Always save the SQL command for future reference or update.

MariaDB follows the SQL standard by the slight differences related to the data types. **The proper definition of data types is critical to database optimizations** regarding the storage capacity and processing speed. For example, defining a Post Code data as CHAR (10) type, it is known that it will have five fixed characters that will result in five unused spaces in the database. While this space seems to be minimal, it becomes critical when thousands or millions of Postal Codes must be stored in the database. MariaDB has many different categories of data types. The most important are numeric, text, and date/time data types. A more detailed analysis may be seen in the MariaDB reference manual.[1]

Numeric data types, in Table 3.4, are used when business data must be stored in terms of numbers, for example, days in production, interest rates, or payroll data. First, the significance of decimal places must be simplified. If not significant, then one of the four INT (integer) types could be selected.

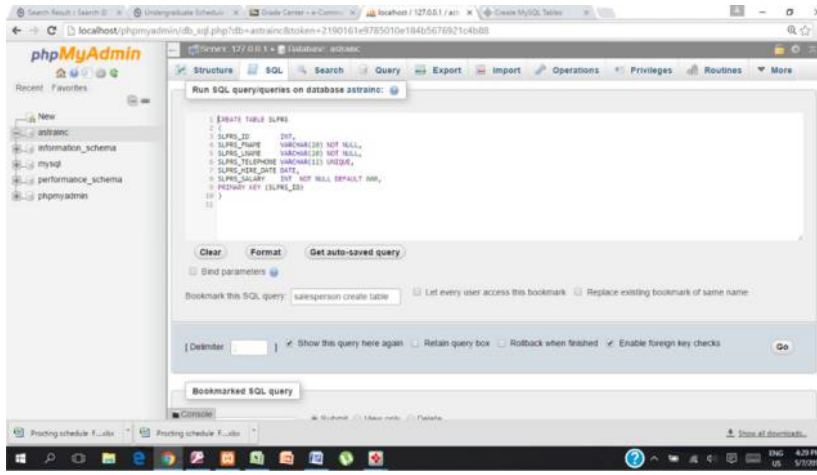


Figure 3.2. Table Creation in MariaDB Using SQL Command Editor.
https://youtu.be/lZYx6k_kY-0

[1] http://dev.mysql.com/doc/refman/5.7/en/data-types.html

Table 3.4. MariaDB Numeric Data Types.

INT	Signed is from −2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. Width can be specified up to 11 digits, for example, INT(4)
TINYINT	If signed, the allowable range is from −128 to 127. If unsigned, the allowable range is from 0 to 255. The width can be specified up to four digits
SMALLINT	If signed, the allowable range is from −32768 to 32767. If unsigned, the allowable range is from 0 to 65535. The width can be specified up to five digits
MEDIUMINT	If signed, the allowable range is from −8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to nine digits
BIGINT	If signed, the allowable range is from −9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. The width can be specified up to 20 digits
FLOAT(M,D)	A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10.2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT
DOUBLE(M,D) Or DECIMAL(M,D)	A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16.4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE

The selection is based optimally on the range of data values the integer may store. In addition, the length could be decided by whether or not a signed value is needed. For example, number of overtime hours could not be negative. The selection is based on the less space consuming data type that can hold the largest possible range of values.

When the specification requires a decimal part of a number, then the precision of decimal places is important, especially in financial indicators or currencies. Different precision ranges may produce different results when decimal numbers are used in arithmetic formulas and calculations. When a more than 24 decimal place decision is required, then double data type is preferred.

Text data types, as in Table 3.5, are used when data will not be used in numeric calculations. Text can be any set of numbers, text characters (multilingual), and special characters like @#$%. The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained.

The string basic types are CHAR, VARCHAR, BLOB, TEXT, and ENUM. Usually, VARCHAR is used because it is more efficient in storing and retrieving variable-length strings. A *BLOB **is a large** binary that contains a variable amount of data. Values as pictures or archives are treated as binary strings. Text is treated as a string of characters.

The date and time types, in Table 3.6, represent DATE, TIME, DATETIME, TIMESTAMP, and YEAR. Each type has a range of valid values, as well as a "zero" value.

MariaDB explains abbreviated TIME values with colons as the time of the day. Suppose "09:10" means "09:10:00," not "00:09:10." MariaDB understands the abbreviated values without colons as that; the two rightmost digits represent seconds. For example, we think of "0910" and 0910 as meaning "09:10:00", that is, 10 minutes after 9 o'clock but the reality is MariaDB understands them as "00:09:10", that is, 9 minutes and 10 seconds. So, be careful about using abbreviated time in MariaDB. By default, the values of time that lie outside the TIME are converted to the valid range of time values. For example, "-930:00:00" and "930:00:00" are converted to "-838:59:59" and "838:59:59." Invalid TIME values

Table 3.5. MariaDB Text Types.

VARCHAR(M)	A variable-length string between 1 and 255 characters in length, for example, VARCHAR (25). The allocated varies to the length of the data store. For example, if VARCHAR (10) is specified and the data stored is "hello" that will occupy only 5 out of 10 characters
CHAR(M)	A fixed-length string between 1 and 255 characters in length (e.g., CHAR (5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1. It does not vary and occupies all specified space regardless of the size of data stored
ENUM	When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C," you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field
TEXT OR BLOB*	MEDIUMTEXT OR MEDIUMBLOB Maximum length of 16777215 characters. LONGTEXT OR LONGBLOB Maximum length of 4294967295 characters

Table 3.6. MariaDB Date and Time Data Types.

DATE	A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30
DATETIME	A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30, 1973 would be stored as 1973-12-30 15:30:00
TIMESTAMP	A timestamp between midnight, January 1, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS). Values are converted from the current time zone to UTC while storing, and converted back from UTC to the current time zone when retrieved
TIME	Stores the time in HH:MM:SS format
YEAR(M)	Stores a year in two-digit or four-digit format. If the length is specified as 2 (e.g., YEAR(2)), YEAR can be 1970 to 2069 (70 to 69). If the length is specified as 4, YEAR can be 1901 to 2155. The default length is 4

are converted to "00:00:00," because "00:00:00" is itself a valid TIME value in MariaDB.

The YEAR type is a 1-byte type used to represent year values. It can be declared as YEAR(2)/YEAR (2) or YEAR(4)/YEAR (4) to specify a display width of two or four characters. If no width is given, the default is four characters YEAR(4)/YEAR (4) and YEAR(2)/YEAR (2), which have different display formats, but have the same range of values.

The DATETIME type is used when you need values containing both date and time information. MariaDB retrieves and displays DATETIME values in "YYYY-MM-DD HH:MM:SS" format. The supported range is "1000-01-01 00:00:00" to "9999-12-31 23:59:59." The TIMESTAMP data type is also used when you need values containing both date and time information. TIMESTAMP has a range of "1970-01-01 00:00:01" UTC to "2038-01-19 03:14:07" UTC[2] (see Figure 3.3).

The major difference between DATETIME and TIMESTAMP is that TIMESTAMP values are converted from the current time zone to UTC while

[2]UTC – Coordinated Universal Time; www.worldtimeserver.com/current_time_in_UTC.aspx

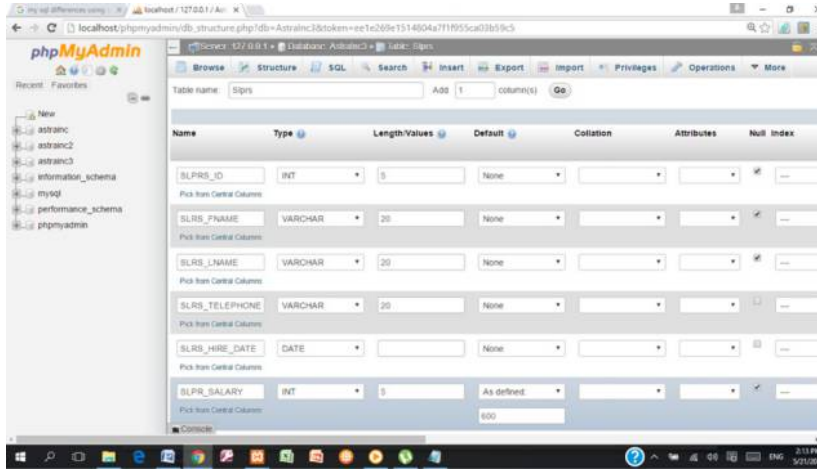


Figure 3.3. Visual Table Creation Using MariaDB Visual Environment.
https://youtu.be/CrpjCqSTWxo

storing, and converted back from UTC to the current time zone when retrieved. The datetime data type value is unchanged.

Creating Tables in MariaDB Visually. In the visual environment of my SQL, table creation can be done without a novice user typing the SQL command for table creation. The creation of tables using the MariaDB visual environment follows the video presentations (Figures 3.2 and 3.3). The resulting database is the same with the database that had been created using the SQL command editor in the previous section.

3.2.1.2. Creating Tables in ORACLE

The creation of database tables in ORACLE could be implemented using two methods:

(1) Direct execution of CREATE TABLE SQL command through the SQL command editor, or
(2) Visual creation of database tables through ORACLE Express interactive environment.

This chapter examines both methods; students may use the manual method to create a number of tables and the visual method to create other tables.

Creating Tables in ORACLE Express with the SQL Command Editor. Open ORACLE Express and insert in the SQL window, figure the SQL command for table (see Figure 3.4 and Table 3.7).

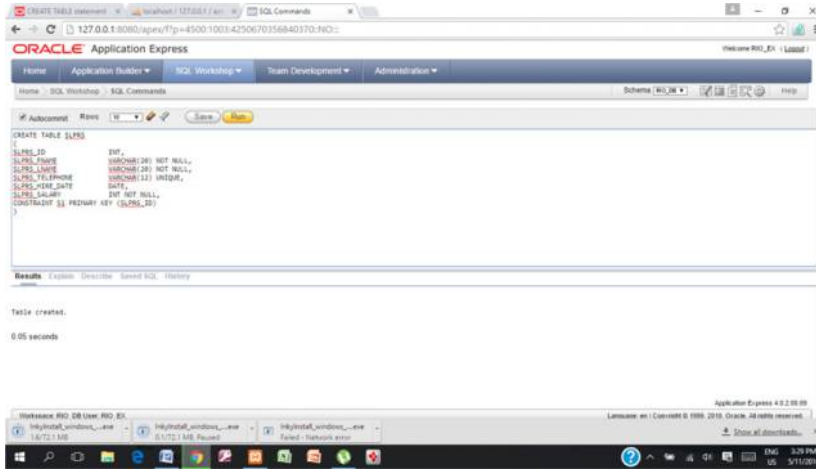


Figure 3.4. Table Creation in ORACLE Express Using the SQL Command Editor.
https://www.youtube.com/watch?v=C5O_hE5MIqE

Table 3.7. ORACLE Numeric Data Types.

NUMBER [Precision, Scale]	Number having precision p and scale s. The precision p can range from 1 to 38. The scales can range from −84 to 127. It requires from 1 to 22 bytes
FLOAT(n)	The number n indicates the number of bits of precision that the value can store. The value for n can range from 1 to 126
BINARY_FLOAT	32-bit floating-point number. This data type requires 5 bytes, including the length byte
BINARY_DOUBLE	64-bit floating-point number. This data type requires 9 bytes, including the length byte

NUMBER data type is significant and often used in string numeric values. Number data types are specified by precision and scale.

- p is the precision, or the total number of significant decimal digits, where the most significant digit is the left-most nonzero digit, and the least significant digit is the rightmost known digit. Oracle guarantees the portability of numbers with precision of up to 20 base-100 digits, which is equivalent to 39 or 40 decimal digits depending on the position of the decimal point.
- s is the scale, or the number of digits from the decimal point to the least significant digit. The scale can range from −84 to 127.

- Positive scale is the number of significant digits to the right of the decimal point to and including the least significant digit.
- Negative scale is the number of significant digits to the left of the decimal point, to but not including the least significant digit. For negative scale, the least significant digit is on the left side of the decimal point, because the actual data are rounded to the specified number of places to the left of the decimal point. For example, a specification of (10,-2) means to round to hundreds (see Table 3.8).

Floating-point numbers can have a decimal point anywhere from the first to the last digit or can have no decimal point at all. Binary floating-point numbers differ from NUMBER in the way the values are stored internally by the Oracle database. Values are stored using decimal precision for NUMBER. All literals that are within the range and precision supported by NUMBER are stored exactly as NUMBER. Literals are stored exactly because literals are expressed using decimal precision (the digits 0 through 9). Binary floating-point numbers are stored using binary precision (the digits 0 and 1). Such a storage scheme cannot represent all values using decimal precision exactly. Frequently, the error that occurs when converting a value from decimal to binary precision is undone when the value is converted back from binary to decimal precision. The literal 0.1 is such an example.

Text data types, as in Table 3.9, are used when data will not be used in numeric calculations. Text can be any set of numbers, text characters (multilingual), and special characters like @#$%. The VARCHAR2 types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained. ORACLE automatically converts VARCHAR to VARCHAR2.

Table 3.8. Examples of Number Data Type in ORACLE.

Actual Data	Specified As	Stored As
123.89	NUMBER	123.89
123.89	NUMBER(3)	124
123.89	NUMBER(6,2)	123.89
123.89	NUMBER(6,1)	123.9
123.89	NUMBER(6,-2)	100
.01234	NUMBER(4,5)	.01234
.00012	NUMBER(4,5)	.00012
.000127	NUMBER(4,5)	.00013
.0000012	NUMBER(2,7)	.0000012
.00000123	NUMBER(2,7)	.0000012

Table 3.9. ORACLE Text Data Types.

VARCHAR2(size [BYTE | CHAR])	Variable-length character string having maximum length size bytes or characters. Maximum size is 4000 bytes or characters, and minimum is 1 byte or 1 character. You must specify size for VARCHAR2. BYTE indicates that the column will have byte length semantics; CHAR indicates that the column will have character semantics
NVARCHAR2(size)	Variable-length Unicode character string having maximum length size characters. The number of bytes can be up to two times the size for AL16UTF16 encoding and three times the size for UTF8 encoding. Maximum size is determined by the national character set definition, with an upper limit of 4000 bytes. You must specify size for NVARCHAR2
CHAR [(size [BYTE | CHAR])]	Fixed-length character data of length size bytes. Maximum size is 2000 bytes or characters. Default and minimum size is 1 byte BYTE and CHAR have the same semantics as for VARCHAR2
NCHAR[(size)]	Fixed-length character data of length size characters. The number of bytes can be up to two times the size for AL16UTF16 encoding and three times the size for UTF8 encoding. Maximum size is determined by the national character set definition, with an upper limit of 2000 bytes. Default and minimum size is 1 character
CLOB	A character large object containing single-byte or multibyte characters. Both fixed-width and variable-width character sets are supported, both using the database character set. Maximum size is (4 gigabytes −1) * (database block size)
NCLOB	A character large object containing Unicode characters. Both fixed-width and variable-width character sets are supported, both using the database national character set. Maximum size is (4 gigabytes −1) * (database block size). Stores national character set data
BLOB	A binary large object. Maximum size is (4 gigabytes −1) * (database block size)
BFILE	Contains a locator to a large binary file stored outside the database. Enables byte stream I/O access to external LOBs residing on the database server. Maximum size is 4 gigabytes

The NVARCHAR2 data type is a Unicode-only data type. When you create a table with an NVARCHAR2 column, you supply the maximum number of characters it can hold. Oracle subsequently stores each value in the column exactly as you specify it, provided the value does not exceed the maximum length of the column.

The maximum length of the column is determined by the national character set definition. Width specifications of character data type NVARCHAR2 refer to the number of characters. The maximum column size allowed is 4,000 bytes. Please refer to Oracle Database Globalization Support Guide[3] for information on Unicode data type support.

The CHAR data type specifies a fixed-length character string. Oracle ensures that all values stored in a CHAR column have the length specified by size. If you insert a value that is shorter than the column length, then Oracle blank-pads the value to column length.

The VARCHAR2 data type specifies a variable-length character string. When you create a VARCHAR2 column, you supply the maximum number of bytes or characters of data that it can hold. Oracle subsequently stores each value in the column exactly as you specify it, provided the value does not exceed the column's maximum length of the column. If you try to insert a value that exceeds the specified length, then Oracle returns an error. The system date and time in ORACLE is referenced as SYSDATE (see Table 3.10).

The DATE data type stores date and time information. For each DATE value, Oracle stores the following information: century, year, month, date, hour, minute, and second.

The TIMESTAMP data type is an extension of the DATE data type. It stores the year, month, and day of the DATE data type, plus hour, minute, and second values.

TIMESTAMP WITH LOCAL TIME ZONE is another variant of TIMESTAMP that includes a time zone offset in its value. It differs from TIMESTAMP WITH TIMEZONE in that data stored in the database is normalized to the database time zone, and the time zone offset is not stored as part of the column data.

When a user retrieves the data, Oracle returns it in the user's local session time zone. The time zone offset is the difference (in hours and minutes) between local time and UTC (Coordinated Universal Time – formerly Greenwich Mean Time). This data type is useful for displaying date information in the time zone of the client system in a two-tier application.

INTERVAL YEAR TO MONTH stores a period of time using the YEAR and MONTH datetime fields. This data type is useful for representing the difference between two datetime values when only the year and month values are significant.

[3] http://docs.oracle.com/cd/B19306_01/server.102/b14225/ch6unicode.htm#NLSPG006

Table 3.10. ORACLE DATE/TIME Data Types.

DATE	This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND
TIMESTAMP [(fractional_seconds)] TIMESTAMP [(fractional_seconds)] WITHTIME ZONE TIMESTAMP [(fractional_seconds)] WITHLOCAL TIME ZONE	Year, month, and day values of date, as well as hour, minute, and second values of time, where fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values of fractional_seconds_precision are 0 to 9. The default is 6. It contains fractional seconds but does not have a time zone. You may select TIMESTAMP with time zone and local time zone
INTERVAL YEAR [(year_precision)] TOMONTH	Stores a period of time in years and months, where year_precision is the number of digits in the YEAR datetime field. Accepted values are 0 to 9. The default is 2. The size is fixed at 5 bytes
INTERVAL DAY [(day_precision)] TO SECOND[(fractional_seconds)]	Stores a period of time in days, hours, minutes, and seconds, where day_precision is the maximum number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2 fractional_seconds_precision is the number of digits in the fractional part of the SECOND field. Accepted values are 0 to 9. The default is 6. The size is fixed at 11 bytes

Creating Tables in ORACLE Visually. In the visual environment of my ORACLE, table creation can be done without a novice user typing the SQL command for table creation (see Figure 3.5).

3.2.1.3. Creating Tables in MS-Access

The creation of database tables in MS-Access could be implemented using two methods:

(1) Direct execution of CREATE SQL command through the SQL command editor, or

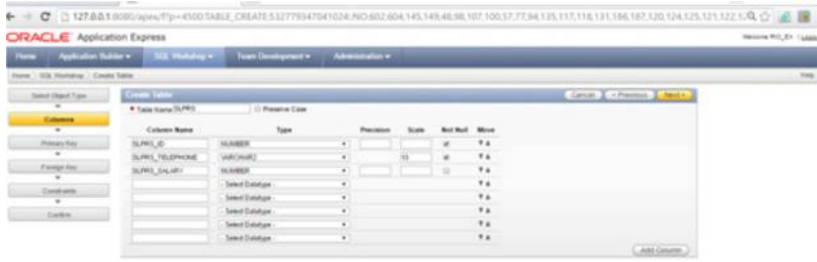


Figure 3.5. Visual Table Creation Using ORACLE Express.
https://youtu.be/-RSbuJkI-hw

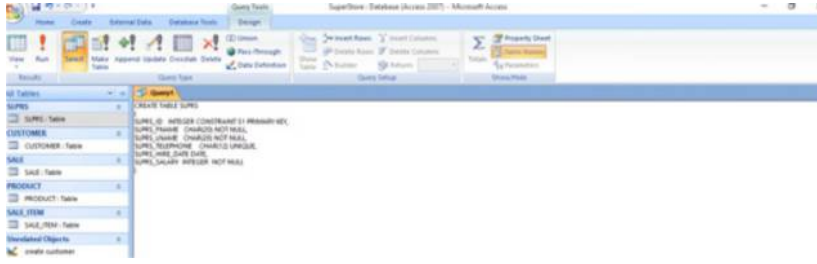


Figure 3.6. Table Creation in MS-Access Using the SQL View.
https://www.youtube.com/watch?v=VmBUef8gsW0&feature=youtube_gdata

(2) Visual creation of database tables through the MS-Access interactive environment.

This chapter examines both methods; students may use the manual method to create a number of tables and the visual method to create other tables.

Creating Tables in MS-Access Using the SQL Command Editor. Open MS-Access, choose Design Menu, and then select Query Design Follow video on (Figure 3.6).

The following MS-Access data types are used.[4] MS_Access data types are slightly different from standard SQL. In Table 3.11 notice the AutoNumber and ReputationID (similar to Sequence in ORACLE), Currency, YesNo (Boolean), OLE for including windows objects, and Hyperlink (storing web links).

[4]https://msdn.microsoft.com/en-us/library/ms714540(v = vs.85).aspx

Table 3.11. MS-Access Data Types.

BYTE	Numbers from 0 to 255 (no decimals). Occupies in byte
INTEGER	Numbers from −32,768 to 32,767 (no decimal). Occupies 2 bytes
LONG INTEGER	(Default) Numbers from −2,147,483,648 to 2,147,483,647 (no decimals)
AUTONUMBER	Unique sequential (incrementing by 1) or random numbers automatically inserted when a record is added
REPLICATION ID	Globally unique identifier (GUID). GUID is a unique reference number used as an identifier in software and could be automatically generated through online GUID generator[5]
DECIMAL	Numbers from −10^28-1 through 10^28-1. Precision 28. Occupies 12 bytes
FLOAT or DOUBLE	They have a precision of up about 7 decimal digits (float) or 15 digits (double). Floats use 4 bytes, and doubles use 8 bytes of disk space
Currency	The currency type is a special kind of decimal, with up to 4 digits on the right of the decimal point and up to 15 on the left. This type uses 8 bytes
YES/NO	A Boolean value that can be either on or off, displayed as a check box in Microsoft Access. Displayed as 0 or 1 in MDB Viewer
OLE	An OLE object field can contain documents, such as images, text files, Microsoft Word files or Microsoft Excel files. The size of this field is limited to 1 GB. You can extract OLE files by dragging the icon to the Finder
TEXT	The Text data type is used for short text fields of limited length, and can contain up to 255 characters
MEMO	A Memo field is a virtually unlimited text field. It can store up to 1 GB of text
DATE/TIME	Specifies a specific point in time (date and time of day). Dates are internally stored as an 8-byte double precision floating-point numbers, so the range is virtually unlimited. (Dates up to 2 million AD can be stored with a precision of 1 second)
HYPERLINK	Field that will store hyperlinks. A hyperlink can be a UNC path or a URL

[5]https://www.guidgenerator.com/

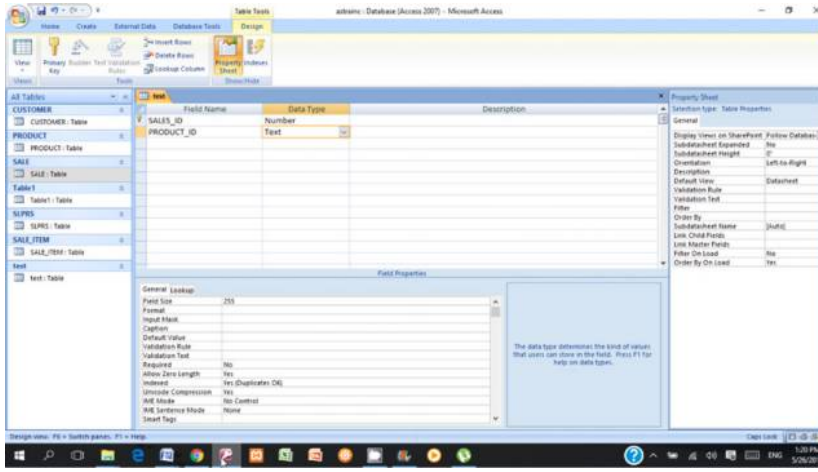


Figure 3.7. Table Creation in MS-Access Visually.
https://www.youtube.com/watch?v=dTpkefRJMCc&feature=youtube_gdata

To create tables using the MS-Access visual environment follow the video on Figure 3.7.

3.3. Inserting Data into the Database

Inserting data is a critical process regarding the accountability of the database. Data insertion must follow quality-designed insertion procedures to avoid erroneous and incomplete resulting information. "Garbage in Garbage out" is a common saying in information science and reflects the low level of usability of produced information when mistakes occur in data entry. Have in mind that databases usually consist of thousands, maybe millions, of records and enormous effort is required to clean a database from erroneous data entries.

Technically, there are several options for entering data into the database. The first option is to use the SQL INSERT command. When inserting into a table, the **number of rows is increased by one.** The SQL INSERT command **inserts a single row of data into a table.** It can be written in two forms:

```
INSERT INTO TableName VALUES (Value1, Value2.....ValueN);
```

or

```
INSERT INTO Tablename (Column1, Column2...ColumnN) VALUES (Value1,
Value2.....ValueN);
```

Assuming the following – The Salesperson Table (SLPRS) is as follows:

SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
120	Rio	Astros	38756448	20/11/15	222,786
281	Sam	Koori	38771990	(NULL)	1090,887
654	Jill	Adams	39881009	23/12/13	356,435
700	Dorian	Lee	36554772	30/5/14	234,411
771	Nick	Manner	(NULL)	(NULL)	436,652
882	Grace	Ramos	33224593	18/8/15	345,523

The SQL command inserts all data into the table matching each data entry with the corresponding table attribute. For example, SLPRS_ID is matched with 921 in the VALUES clause.

The SQL command will insert a new row in table salesperson. Notice, since the attributes are not specified, data for all attributes must be declared. TEXT, DATE data must be inserted using single quotes to distinguish them from numeric values likes SLPRS_SALARY that are entered without a single quote.

In case some of the data values are empty (i.e., we do not have any record of them during data insertion), then place an empty space between comma separators for numeric values and '' for text values, for example:

```
INSERT INTO SLPRS VALUES (921,'Reema', 'AlRashid','37002345',
22/11/2013, 220300);
```

SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
120	Rio	Astros	38756448	20/11/15	222786
281	Sam	Koori	38771990	(NULL)	109887
654	Jill	Adams	39881009	23/12/13	356435
700	Dorian	Lee	36554772	30/5/14	234411
771	Nick	Manner	(NULL)	(NULL)	436652
882	Grace	Ramos	33224593	18/8/15	345523
921	Reema	AlRashid	37002345	22/11/2013	220300

```
INSERT INTO SLPRS VALUES (921,'Reema', 'Arachnid','', , );
```

Notice that certain data for SLPRS_TELEPHONE and SLPRS_HIRE_DATE are empty. That means that **no data** are entered for these fields. In databases, the keyword *NULL* indicates no data entry. When the SLPRS table is created in

the previous sections, all attributes of the table are specified as NOT NULL with exception of the SLPRS_TELEPHONE, SLPRS_SALARY, and SLPRS_HIRE_DATE attributes. Not entering data into these columns will not violate any database constraints.

When partial data must be inserted into a database, then the inserted attributes must be specified in a parenthesis with one to one correspondence with the data declared in the VALUES clause. The SQL command

```
INSERT INTO SLPRS (SLPRS_ID, SLPRS_FNAME, SLPRS_LNAME,
SLPRS_TELEPHONE)
VALUES (925,'Julia', 'Marconi','38899742');
```

SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
120	Rio	Astros	38756448	20/11/15	222786
281	Sam	Koori	38771990	(NULL)	109887
654	Jill	Adams	39881009	23/12/13	356435
700	Dorian	Lee	36554772	30/5/14	234411
771	Nick	Manner	(NULL)	(NULL)	436652
882	Grace	Ramos	33224593	18/8/15	345523
921	Reema	AlRashid	37002345	22/11/2013	220300
925	Julia	Marconi	39480511	(NULL)	600

will insert data for LSPRS_ID, SLPRS_FNAME, SLPRS_LNAME, and SLPRS_TELEPHONE and will not insert any data for the rest of the fields. Notice that although data for SLPRS_SALARY were not entered, the **DEFAULT** value specified during table (600) creating is entered automatically. The order of the attributes may change, if it is necessary, as long as it is one to one correspondence between attributes and the data in the VALUES clause. For example, this SQL command will result in the same insertion as the previous command.

```
INSERT INTO SLPRS (SLPRS_LNAME, SLPRS_FNAME, SLPRS_TELEPHONE,
SLPRS_ID)
VALUES ('Marconi' ,'Julia', '38899742', 925);
```

If the primary key is specified as AUTONUMBER with an automatic increment, then it may be ignored only in the VALUES clause. For example, if SLPRS_ID is declared as AUTONUMBER, then the system assigns a value to the primary key. In this case, the SQL command will insert the next available number for SLPRS_ID.

```
INSERT INTO SLPRS (SLPRS_ID, SLPRS_FNAME, SLPRS_LNAME,
SLPRS_TELEPHONE)
VALUES ('Julia', 'Marconi','38899742');
```

Table 3.12. Common Mistakes in Data Insertion.

No values are entered for a primary key. Primary key values cannot be NULL
`INSERT INTO SLPRS (SLPRS_ID, SLPRS_FNAME, SLPRS_LNAME, SLPRS_TELEPHONE)`
`VALUES ('Julia', 'Marconi','38899742');`
An existing value is entered for a primary key. By default, it is unique
`INSERT INTO SLPRS VALUES (921,'George', 'Reed','38776554', '10/11/2013',140300);`
Primary key *921* already exists
Attributes specified NOT NULL are omitted in data insertion
`INSERT INTO SLPRS (SLPRS_ID, SLPRS_LNAME, SLPRS_TELEPHONE)`
`VALUES (925, 'Marconi','38899742');`
SLPRS_FNAME is specified as NOT NULL; it cannot be omitted
Data types violations for example a text is inserted as number
`INSERT INTO SLPRS VALUES (921,Reema, 'AlRashid','37002345', '22/11/2013','220300');`
SLPRS_FNAME Reema is entered without single quotes, and SLPRS_SALARY is entered as text
A foreign key data value that does not exist as primary value in the table containing the primary key
`INSERT INTO SALE (SALE_ID, SALE_DATE, SLPRS_ID, CUS_ID`
`VALUES (1001, '22/10/2016', 999, 700);`
SLPRS_ID 999 does not exist in the salesperson table

When inserting data into a table, all the constraints that are specified during data creation must be applied. If not, the SQL command is not executed for all data and **returns a NULL** value with an error message. Some common mistakes in data insertion that result in invalid commands are shown in Table 3.12.

Apart from using the INSERT SQL command in SQL VIEW to enter a single row of data into the database, it can be done using the MS-Access visual environment. It is a straightforward action. Again, all constraint validations apply while you are entering the data. When large amounts of data need to be inserted into a database, it is preferable to insert them using external data files. In MS-Access, these data import may take place using data from another Access Database, from MS-Excel, text files, or XML type files (see Figures 3.8–3.10).

3.4. Deleting and Updating Data

When the data of the database are deleted or updated (modified), the database is changing. Careful steps must be taken when these activities are invoked to minimize the risk of losing valuable data.

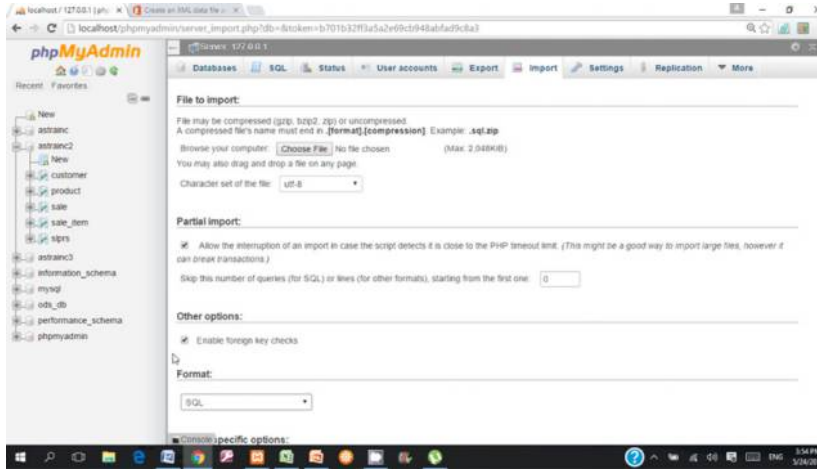


Figure 3.8. Inserting Data in MariaDB.
https://www.youtube.com/watch?v=P7aom45JC_0&feature=youtube_gdata.

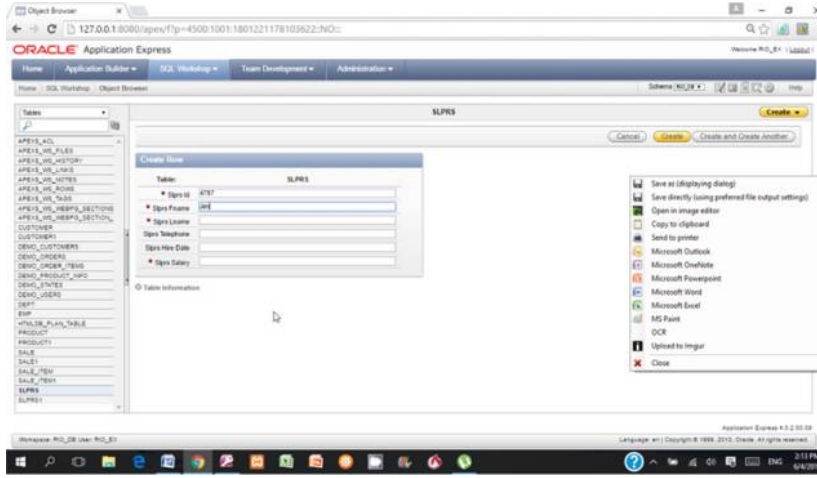


Figure 3.9. Inserting Data in ORACLE.
https://www.youtube.com/watch?v=kbn9bVutY7s&feature=youtube_gdata.

3.4.1. Deleting Data from a Database

Deleting data from a database is common activity. For example, when an employee contract is terminated, then this employee must be deleted from the employee list. When deleting from a table, the **number of rows is reduced**

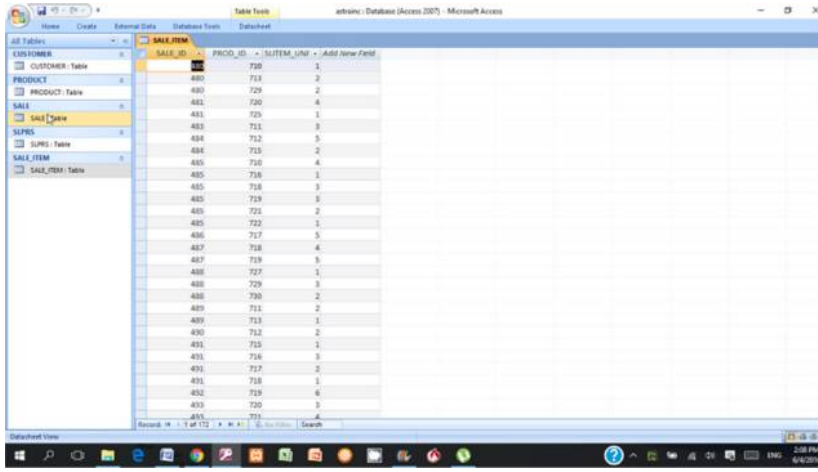


Figure 3.10. Inserting Data in MS-Access.
https://www.youtube.com/watch?v=VmBUef8gsW0&feature=youtube_gdata.

by one or many. To delete data from a database, the DELETE SQL command is used, which may delete **one or many rows or even the entire set of rows** from a table. The syntax for the DELETE SQL command is:

```
DELETE FROM Tablename WHERE some_column=some_value;
```

The command deletes the whole row and not partial fields, for example, the row that has SLPRS_ID = 654 or the rows that SLPRS_HIRE_DATE = NULL (two rows will be deleted). The DELETE command does not delete fields, for example, the Last Name for salesperson 281. In forming the DELETE command, the following conditional operators apply:

=	Checks if the values of two operands are equal or not; if yes then condition becomes true.
!= <>	Checks if the values of two operands are equal or not; if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand; if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand; if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes then condition becomes true.

Some examples of applying the DELETE command into the SLPRS table are:

Row No.	SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
1	120	Rio	Astros	38756448	20/11/15	222786
2	281	Sam	Koori	38771990	(NULL)	109887
3	654	Jill	Adams	39881009	23/12/13	356435
4	700	Dorian	Lee	36554772	30/5/14	234411
5	771	Nick	Manner	(NULL)	(NULL)	436652
6	882	Grace	Ramos	33224593	18/8/15	345523
7	921	Reema	AlRashid	37002345	22/11/2013	220300

DELETE FROM *SLPRS* WHERE *SLPRS_ID = 281*; will delete the second row of the SLPRS table.

DELETE FROM *SLPRS* WHERE *SLPRS_LNAME = 'Lee';* will delete the fourth row of the SLPRS table.

DELETE FROM *SLPRS* WHERE *SLPRS_LNAME <> 'Lee';* will delete all rows except the fourth row of the SLPRS table.

DELETE FROM *SLPRS* WHERE *SLPRS_SALARY > 400000;* will delete the fifth row of the SLPRS table.

DELETE FROM *SLPRS* WHERE *SLPRS_SALARY < 356435;* will delete all rows but third and fifth of the SLPRS table. Row 3 is not deleted because the SALARY = 356435. Therefore, the condition returns false for this row.

DELETE FROM *SLPRS* WHERE *SLPRS_SALARY <=356435;* will all rows but the fifth of the SLPRS table.

DELETE FROM *SLPRS* WHERE SLPRS_*HIRE_DATE* <= '*31/12/2013*' will delete the third and seventh rows of the SLPRS table with hire dates previous to 31/12/2013. Notice that the second and fifth rows do not comply with this condition because they have no values with which to compare. Note that quotes are needed for date data types.

In the DELETE command, the following logical operators may be applied in conjunction with the conditional operators.

AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

(*Continued*)

LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used, for example, NOT BETWEEN, NOT NULL, etc. This is a negate operator.

Some examples of using the logical operators in the DELETE SQL command are:

DELETE FROM *SLPRS* WHERE *SLPRS_SALARY* < *400000* AND *SLPRS_SALARY* > *300000* ; or

DELETE FROM *SLPRS* WHERE *SLPRS_SALARY* BETWEEN *400000* AND *300000* ; will delete the third and sixth rows of the SLPRS table with salaries between 300,000 and 400,000.

DELETE FROM *SLPRS* WHERE *SLPRS_SALARY* NOT BETWEEN *400000* AND *300000* ; will delete all rows with salaries between 300,000 and 400,000. The third and sixth rows of the SLPRS table will not be deleted.

DELETE FROM *SLPRS* WHERE *SLPRS_HIRE_DATE* IS NULL; will delete the second and fifth rows of the SLPRS table.

DELETE FROM *SLPRS* WHERE *SLPRS_HIRE_DATE* IS NOT NULL; will delete all rows except the second and fifth rows of the SLPRS table.

DELETE FROM *SLPRS* WHERE *SLPRS_TELEPHONE* IS NULL AND *SLPRS_HIRE_DATE* IS NULL; will delete the fifth row of the SLPRS table because both IS NULL conditions are satisfied.

DELETE FROM *SLPRS* WHERE *SLPRS_TELEPHONE* IS NULL OR *SLPRS_HIRE_DATE* IS NULL; will delete the second and fifth row of the SLPRS table because either of the IS NULL conditions are satisfied.

The **LIKE** logical operator is used when the search relates to parts of the text field. The following wildcard characters are applied with the LIKE logical operator **NOT THE = OPERATOR.**

The percent sign %	Matches one or more characters. Note that MS-Access uses the asterisk (*) wildcard character instead of the percent sign (%) wildcard character.
The underscore _	Matches one character. Note that MS-Access uses a question mark (?) instead of the underscore (_) to match any one character.

DELETE FROM *SLPRS* WHERE *SLPRS_FNAME* LIKE 'R%'; will delete the first and the seventh rows of the SLPRS table that start with the capital letter 'R'.

DELETE FROM *SLPRS* WHERE *SLPRS_LNAME* LIKE '——s%'; will delete the third, sixth, and the seventh rows of the SLPRS table that their last name's fifth character is 's'.

DELETE FROM *SLPRS* WHERE *SLPRS_FNAME* LIKE '?i*'; will delete the first, third, and fifth rows of the SLPRS table where their second character of the first name is 'i' (MS-Access).

Warning: If the WHERE clause is omitted from the DELETE command.

When a DELETE command is applied without a where clause, **ALL the rows** of the table will be deleted, for example:

DELETE FROM *SLPRS;* will result in an empty table.

SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY

This does not mean that the table is deleted but all data rows are deleted. The table structure remains available to insert new data. To complete delete a table (data and structure), the SQL command DROP is used.

DROP TABLE *SLPRS;*

Dropping a table can be hazardous in loosing valuable data and may be done under careful quality procedures. Usually, only the database or root administrator is privileged to delete a table. More of these quality assurance options will be seen in Chapter 10, related to database security.

3.4.2. Updating the Data of a Database

Updating or modifying the data of a database means that certain fields of a row or rows will change. When updating a table the **number of rows remains the same**. Only the data values inside the fields of the rows may change. The following SQL command is used to change data.

UPDATE *Table name* SET *column1 = value1, column2 = value2,...* WHERE *some_column = some_value;*

Examples of using the UPDATE command follow. Again, the same conditional and logical operators with the DELETE command are applied.

Row No.	SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
1	120	Rio	Astros	38756448	20/11/15	222786
2	281	Sam	Koori	38771990	(NULL)	109887
3	654	Jill	Adams	39881009	23/12/13	356435
4	700	Dorian	Lee	36554772	30/5/14	234411

(*Continued*)

Row No.	SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
5	771	Nick	Manner	(NULL)	(NULL)	436652
6	882	Grace	Ramos	33224593	18/8/15	345523
7	921	Reema	AlRashid	37002345	22/11/2013	220300

UPDATE *SLPRS* SET *SLPRS_FNAME= 'James'* WHERE *SLPRS_ID= 771;* will modify the first name of fifth row from 'Nick' to 'James'.

UPDATE *SLPRS* SET *SLPRS_FNAME= 'James', SLPRS_LNAME= 'Hokins'* WHERE *SLPRS_ID= 771;* will change the first name of fifth row from 'Nick' to 'James' and the last name from 'Manner' to 'Hokins'.

UPDATE *SLPRS* SET *SLPRS_TELEPHONE= '33333333'* WHERE *SLPRS_TELEPHONE IS NULL;* will change the telephone of fifth row from NULL to 33333333.

UPDATE *SLPRS* SET *SLPRS_SALARY= 210000* WHERE *SLPRS_HIRE_DATE IS NULL;* will change the salary of the second and fifth rows from NULL to 100,000.

UPDATE *SLPRS SET SLPRS_SALARY= 270000* WHERE *SLPRS_SALARY BETWEEN 200000 AND 250000;* will change the salary of the first, fourth, and seventh rows to 270,000.

UPDATE *SLPRS* SET *SLPRS_FNAME= 'Ryan'* WHERE *SLPRS_FNAME* LIKE 'R%'; will change the first name of the first and the seventh to Ryan.

Warning: If the WHERE clause is omitted from the UPDATE command.

UPDATE *SLPRS* SET *SLPRS_SALARY= 200000;* will change the salary of **all** salespersons to 200,000. When a *WHERE* clause is not included, the changes are universal/catholic to all the rows specified in the *SET* clause.

3.4.3. ON DELETE and ON UPDATE Referential Integrity Constraints

The integrity constraints CASCADE/RESTRICT/SET NULL/SET DEFAULT **are applied to foreign keys** and are included in the CREATE TABLE SQL command. To understand better how integrity constraints operate, consider an example with salesperson table (SLPRS) which is with a 1:M relationship with the SALE table. The primary key of SLPRS (SLPRS_ID) is a foreign key in the SALE table. In turn, the SALE table is in a 1:M relationship with the SALES_ITEM. The primary key of the SALE table (SALE_ID) is a foreign key in the SALE_ITEM table. The problem begins when a primary key is deleted, for example, SLPRS_ID = 307, and leaves occurrences of this key in the tables appearing as a foreign key, named as "orphan child."

SLPRS

Row No	SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
1	120	Rio	Astros	38756448	20/11/15	222786
2	281	Sam	Koori	38771990	(NULL)	109887
3	654	Jill	Adams	39881009	23/12/13	356435
4	700	Dorian	Lee	36554772	30/5/14	234411

SALE

Row No	SALE_ID (PK)	SALE_DATE	SLPRS_ID	CUST_ID
1	300	30/10/2015	120	507
2	301	2/11/2015	654	568
3	302	09/07/2015	281	307
4	303	26/11/2015	120	734
5	304	15/12/2015	307	658

SALE_ITEM

Row No	SALE_ID (PK)	PROD_ID	SLITEM_NUMBER
1	300	700	2
2	300	820	5
3	301	677	3
4	302	700	4
5	303	347	2

The following CREATE Table SQL are assumed for the creation of the three tables. The foreign keys optionally might be set with ON DELETE or/and ON CASCADE constraints.

```
CREATE TABLE SLPRS
(
SLPRS_IDINT,
SLPRS_FNAME VARCHAR(20) NOT NULL,
SLPRS_LNAME VARCHAR(20) NOT NULL,
```

```
CREATE TABLE SALE
(
SALE_ID INTEGER CONSTRAINT SL1 PRIMARY KEY,
SALE_DATE DATE NOT NULL,
SLPRS_ID INTEGER NOT NULL,
CUST_ID INTEGER NOT NULL,
```

```
CREATE TABLE SALE_ITEM
(
SALE_ID INTEGER,
PROD_ID INTEGER,
SLITEM_UNITS INTEGER NOT NULL,
CONSTRAINT SLI1 PRIMARY KEY (SALE_ID, PROD_ID),
CONSTRAINT SL5 FOREIGN KEY (SALE_ID) REFERENCES
```

(*Continued*)

SLPRS_TELEPHONE VARCHAR(12) UNIQUE, SLPRS_HIRE_DATE DATE, SLPRS_SALARY INT NOT NULL, CONSTRAINT S1 PRIMARY KEY (SLPRS_ID))	CONSTRAINT SL2 FOREIGN KEY (SLPRS_ID) REFERENCES SLPRS (SLPRS_ID) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT SL3 FOREIGN KEY (CUST_ID) REFERENCES CUSTOMER(CUST_ID))	SALE(SALE_ID) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT SL6 FOREIGN KEY (PROD_ID) REFERENCES PRODUCT (PROD_ID))

The options that the ON DELETE or ON UPDATE constraints might take are:

CASCADE	allows the successive deletion or update of foreign keys
RESTRICT	does allow the deletion of foreign keys
SET NULL	updates the foreign key to NULL
SET DEFAULT	updates the foreign key to a DEFAULT value

The role of these constraints is to ensure **referential integrity**, which means that the relationships between tables remain consistent (Pratt & Adamski, 2012); data in foreign keys must be consistent with the data in primary keys. Since by definition a foreign key is referenced to a primary key in another table, this key must exist in the primary table. We noticed that in the SALE table, all referenced foreign key data for SLPRS_ID (120, 281, 654) are existing in the primary key SLPRS_ID in the SLPRS table. For example, SLPRS_ID 307 does not exist in the SLPPR table. That creates a violation of referential integrity since the foreign contains a nonexistent value in the referenced table, which is called orphan child.

The inconsistency/incontinency (unrestrained flow) relies on the fact that there is no data for salesperson 307. In case the value was NULL instead of 307, the nonaccountability of salesperson information for sale 304 does not constitute a referential integration violation. In any case, NULL values are not allowed by the NOT NULL restriction for the SLPRS_ID column in table SALE.

Assuming the above table creation constraints, an attempt may be made to delete salesperson with SLPRS = 120.

```
DELETE FROM SLPRS WHERE SLPRS_ID= 120;
```

This command will delete the salesperson with ID = 120 in row 1 of the SLPRS table. The ON DELETE CASCADE constraint on the SALE will delete all occurrences of SLPRS_ID = 120 in the Sale table. Therefore, rows 1 and 4 with SALE_ID equal to 301 and 303 will automatically deleted. In turn, the ON

DELETE CASCADE constraint on the SALE_ITEM table will delete rows 1, 2, and 5 where the foreign key SALE_ID has values 300 or 303. This effect actually is a disaster chain. The resulting tables are presented below:

SLPRS

Row No	SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
1	281	Sam	Koori	38771990	(NULL)	109887
2	654	Jill	Adams	39881009	23/12/13	356435
3	700	Dorian	Lee	36554772	30/5/14	234411

SALE

Row No	SALE_ID (PK)	SALE_DATE	SLPRS_ID	CUST_ID
1	301	2/11/2015	654	568
2	302	09/07/2015	281	307

SALE_ITEM

Row No	SALE_ID (PK)	PROD_ID	SLITEM_NUMBER
1	301	677	3
2	302	700	4

In the case where the ON DELETE CASCADE constraint was not enforced in both SALE and SALE_ITEM tables, the salesperson with 120 will be deleted from the SLPRS table but the sales that have been by this salesperson will remain in the SALES table, as orphan child, and also in the SALE_ITEM table.

```
CREATE TABLE SLPRS
(
SLPRS_IDINT,
SLPRS_FNAME VARCHAR(20) NOT NULL,
SLPRS_LNAME VARCHAR(20) NOT NULL,
SLPRS_TELEPHONE VARCHAR(12) UNIQUE,
```

```
CREATE TABLE SALE
(
SALE_ID INTEGER CONSTRAINT SL1 PRIMARY KEY,
SALE_DATE DATE NOT NULL,
SLPRS_ID INTEGER NOT NULL,
CUST_ID INTEGER NOT NULL,
CONSTRAINT SL2 FOREIGN KEY (SLPRS_ID) REFERENCES SLPRS (SLPRS_ID) CONSTRAINT SL3 FOREIGN KEY (CUST_ID)
```

```
CREATE TABLE SALE_ITEM
(
SALE_ID INTEGER,
PROD_ID INTEGER,
SLITEM_UNITS INTEGER NOT NULL,
CONSTRAINT SLI1 PRIMARY KEY (SALE_ID, PROD_ID),
CONSTRAINT SL5 FOREIGN KEY (SALE_ID) REFERENCES SALE(SALE_ID) CONSTRAINT SL6 FOREIGN KEY (PROD_ID) REFERENCES PRODUCT (PROD_ID)
```

(*Continued*)

SLPRS_HIRE_DATE DATE, SLPRS_SALARY INT NOT NULL, CONSTRAINT S1 PRIMARY KEY (SLPRS_ID))	REFERENCES CUSTOMER (CUST_ID))	)

SLPRS

Row No	SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
1	281	Sam	Koori	38771990	(NULL)	109887
2	654	Jill	Adams	39881009	23/12/13	356435
3	700	Dorian	Lee	36554772	30/5/14	234411

SALE

Row No	SALE_ID (PK)	SALE_DATE	SLPRS_ID	CUST_ID
1	300	30/10/2015	120	507
2	301	2/11/2015	654	568
3	302	09/07/2015	281	307
4	303	26/11/2015	120	734
5	304	15/12/2015	307	658

SALE_ITEM

Row No	SALE_ID (PK)	PROD_ID	SLITEM_NUMBER
1	300	700	2
2	300	820	5
3	301	677	3
4	302	700	4
5	303	347	2

If the ON DELETE CASCADE constraint was enforced in the SALE table but not in the SALE_ITEM table, then the deletion of salesperson 120 will result in deletion of SALE items sales **but SALE_ITEM rows** 1, 2, and 5 **will not be deleted**.

SLPRS

Row No	SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
1	281	Sam	Koori	38771990	(NULL)	109887
2	654	Jill	Adams	39881009	23/12/13	356435
3	700	Dorian	Lee	36554772	30/5/14	234411

SALE

Row No	SALE_ID (PK)	SALE_DATE	SLPRS_ID	CUST_ID
1	301	2/11/2015	654	568
2	302	09/07/2015	281	307

SALE_ITEM

Row No	SALE_ID (PK)	PROD_ID	SLITEM_NUMBER
1	300	700	2
2	300	820	5
3	301	677	3
4	302	700	4
5	303	347	2

The ON UPDATE CASCADE constraint has similar effects to the ON DELETE CASCADE constraint. Assume that salesperson with SLPRS_ID = 120 must be changed to SLPRS_ID 150 and SALE_ID with ID 301 must be changed to 400.

```
UPDATE SLPRS SET SLPRS_ID= 150 WHERE SLPRS_ID= 120;
UPDATE SALE SET SALE_ID= 301 WHERE SALE_ID= 400;
```

In this example, ON UPDATE CASCADE constraint is enforced in the SLSPR_ID foreign key of the SALE table but not on the SALE_ID foreign key of the SALE_ITEM table. This will result in an orphan child in the SALE_ITEM table, since the SALE_ID is not updated automatically by an ON UPDATE cascade and remains 301.

```
CREATE TABLE SLPRS
(
SLPRS_IDINT,
SLPRS_FNAME VARCHAR(20) NOT NULL,
SLPRS_LNAME VARCHAR(20) NOT NULL,
SLPRS_TELEPHONE VARCHAR(12) UNIQUE,
SLPRS_HIRE_DATE DATE,
SLPRS_SALARY INT NOT NULL,
CONSTRAINT S1 PRIMARY KEY (SLPRS_ID)
)
```

```
CREATE TABLE SALE
(
SALE_ID INTEGER CONSTRAINT SL1 PRIMARY KEY,
SALE_DATE DATE NOT NULL,
SLPRS_ID INTEGER NOT NULL,
CUST_ID INTEGER NOT NULL,
CONSTRAINT SL2 FOREIGN KEY (SLPRS_ID) REFERENCES SLPRS (SLPRS_ID) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT SL3 FOREIGN KEY (CUST_ID) REFERENCES CUSTOMER(CUST_ID)
)
```

```
CREATE TABLE SALE_ITEM
(
SALE_ID INTEGER,
PROD_ID INTEGER,
SLITEM_UNITS INTEGER NOT NULL,
CONSTRAINT SLI1 PRIMARY KEY (SALE_ID, PROD_ID),
CONSTRAINT SL5 FOREIGN KEY (SALE_ID) REFERENCES SALE(SALE_ID) ON DELETE CASCADE,
CONSTRAINT SL6 FOREIGN KEY (PROD_ID) REFERENCES PRODUCT (PROD_ID)
)
```

SLPRS

Row No	SLPRS_ID (PK)	SLPRS_ FNAME	SLPRS_ LNAME	SLPRS_ TELEPHONE	SLPRS_ HIRE_DATE	SLPRS_ SALARY
1	150	Rio	Astros	38756448	20/11/15	222786
2	281	Sam	Koori	38771990	(NULL)	109887
3	654	Jill	Adams	39881009	23/12/13	356435
4	700	Dorian	Lee	36554772	30/5/14	234411

SALE

Row No	SALE_ID (PK)	SALE_DATE	SLPRS_ID	CUST_ID
1	300	30/10/2015	150	507
2	400	2/11/2015	654	568
3	302	09/07/2015	281	307
4	303	26/11/2015	150	734
5	304	15/12/2015	307	658

SALE_ITEM

Row No	SALE_ID (PK)	PROD_ID	SLITEM_NUMBER
1	300	700	2
2	300	820	5
3	301	677	3
4	302	700	4
5	303	347	2

ON DELETE RESTRICT and ON UPDATE RESTRICT constraints do not allow the deletion or update of the primary key when this constraint exists in a foreign key constraint declaration. Both of these commands will return an **error message** when the following constraints are applied.

```
DELETE FROM SLPRS WHERE SLPRS_ID= 120;
UPDATE SLPRS SET SLPRS_ID= 150 WHERE SLPRS_ID=120;
UPDATE SALE SET SALE_ID= 301 WHERE SALE_ID=400;
```

ERROR: Cannot delete or update a parent row, a foreign key constraint fails

```
CREATE TABLE SLPRS
(
SLPRS_IDINT,
SLPRS_FNAME VARCHAR(20) NOT NULL,
SLPRS_LNAME VARCHAR(20) NOT NULL,
SLPRS_TELEPHONE VARCHAR(12) UNIQUE,
SLPRS_HIRE_DATE DATE,
SLPRS_SALARY INT NOT NULL,
CONSTRAINT S1 PRIMARY KEY (SLPRS_ID)
)
```

```
CREATE TABLE SALE
(
SALE_ID INTEGER CONSTRAINT SL1 PRIMARY KEY,
SALE_DATE DATE NOT NULL,
SLPRS_ID INTEGER NOT NULL,
CUST_ID INTEGER NOT NULL,
CONSTRAINT SL2 FOREIGN KEY (SLPRS_ID) REFERENCES SLPRS (SLPRS_ID) ON DELETE RESTRICT ON UPDATE RESTRICT,
CONSTRAINT SL3 FOREIGN KEY (CUST_ID) REFERENCES CUSTOMER(CUST_ID)
)
```

```
CREATE TABLE SALE_ITEM
(
SALE_ID INTEGER,
PROD_ID INTEGER,
SLITEM_UNITS INTEGER NOT NULL,
CONSTRAINT SLI1 PRIMARY KEY (SALE_ID, PROD_ID),
CONSTRAINT SL5 FOREIGN KEY (SALE_ID) REFERENCES SALE(SALE_ID) ON DELETE RESTRICT ON UPDATE RESTRICT,
CONSTRAINT SL6 FOREIGN KEY (PROD_ID) REFERENCES PRODUCT (PROD_ID)
)
```

In case a row must be deleted or updated when the ON DELETE RESTRICT and ON UPDATE RESTRICT are applied on the SALES and SALES_ITEM tables:

- First, the row(s) in the SALE_ITEM table must be deleted or updated.
- Then, the row(s) in the SALE table must be deleted or updated.
- Last, the SLPRS_ID primary key in the SLPRS table must be deleted or updated, if necessary.

The ON DELETE SET NULL and ON UPDATE SET NULL and the ON DELETE SET DEFAULT and ON UPDATE SET DEFAULT constraints are used when a NULL or the predefined DEFAULT value must replace the values of foreign keys that are under deletion or update. The ON DELETE and ON UPDATE constraints can be applied in MariaDB (video 3.10) and ORACLE (video 3.11), either through the SQL command editor included in the CREATE TABLE command or visually. **MS-Access does not allow these constraints in the command level**. The constraints may only be enforced visually by right clicking the relationship lines. See Figures 3.11 and 3.13.

3.4.4. Updating the Table Structure

The *ALTER TABLE* SQL command is to modify the structure column definitions of a table. Some examples are:

`ALTER TABLE SLPRS ADD SLPRS_ADDRESS VARCHAR (30);` will add a new attribute SLPRS_ADDRESS to table SLPRS.
`ALTER TABLE SLPRS DROP SLPRS_TELEPHONE;` will REMOVE attribute SLPRS_TELEPHONE from the table SLPRS.
`ALTER TABLE SLPRS ALTER COLUMN SLPRS_TELEPHONE VARCHAR (50);` will CHANGE attribute SLPRS_TELEPHONE data type to VRCHAR (50) in MS-Access.
`ALTER TABLE SLPRS MODIFY COLUMN SLPRS_TELEPHONE VARCHAR (50);` will CHANGE attribute SLPRS_TELEPHONE data type to VRCHAR (50) in MariaDB.
`ALTER TABLE SLPRS MODIFY SLPRS_TELEPHONE VARCHAR (50);` will CHANGE attribute SLPRS_TELEPHONE data type to VRCHAR (50) in ORACLE 10g or later versions.

In general, CREATE TABLE and ALTER TABLE privileges in a database management system must be carefully granted. Usually, only database administrators have the right to make some modifications to the database (see Figures 3.11–3.13).

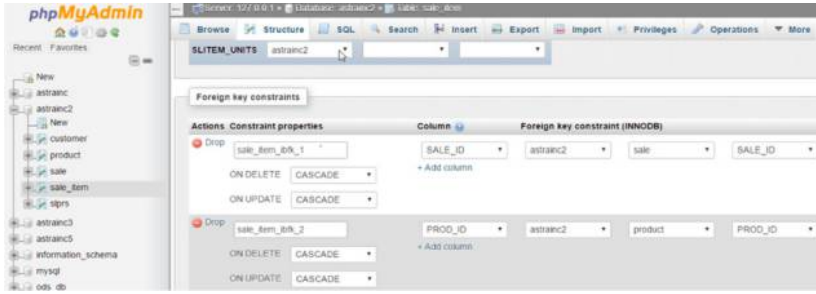


Figure 3.11. Deleting/Updating Data ON DELETE/ON UPDATE CASCADE/ RESTRICT in MariaDB.
https://www.youtube.com/watch?v=VjoOrure7bA&feature=youtube_gdata

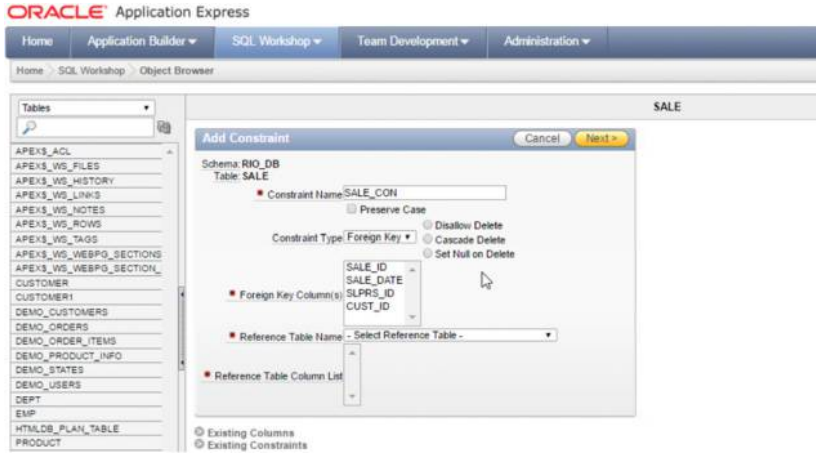


Figure 3.12. Deleting/Updating Data ON DELETE/ON UPDATE CASCADE/ RESTRICT in ORACLE.
https://www.youtube.com/watch?v=CVwlyH4aYUs&feature=youtube_gdata

Summary

SQL or sequel is command language that enables the database users to create, manipulate data, and extract information. Moreover, it is a powerful tool in the hands of managers familiar with SQL to access company data and produce on-demand information to support decision-making.

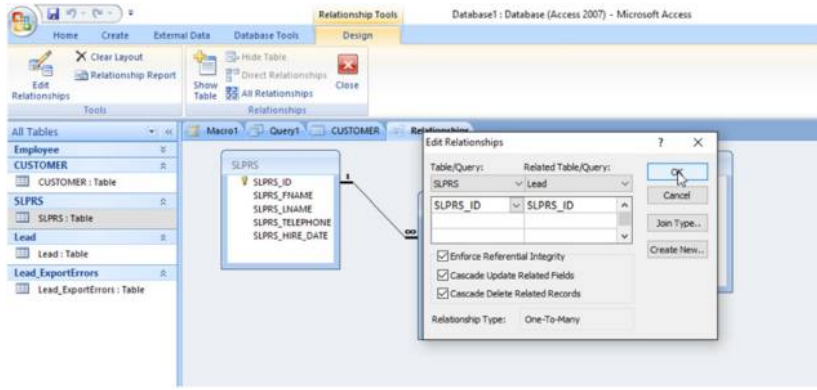


Figure 3.13. Deleting/Updating Data ON DELETE/ON UPDATE CASCADE/ RESTRICT in MS-Access.

https://www.youtube.com/watch?v=mTYwpIRpFCE&feature=youtube_gdata

The creation of database in RDBMS involves three important steps, that is, the creation of the database, the creation of database tables, and the creation of the relationships between these tables. Once the database is created, the next task is to create the structure of the tables using the CREATE TABLE SQL command. Constraints are limitations that the database designers set on the data sets during table creation. Every table must be specified with a primary key constraint that will uniquely identify each row or tulle of the data and must be specified inside the CREATE TABLE command.

The primary key constraint specifies which attribute or attributes, in case of a composite primary key, will be set as a unique row identifier. The Foreign key constraint sets the relationship between primary and foreign keys in 1:M relationships. A table must have one primary key and zero, one or many foreign keys. The basic data types are referring to numeric, text, and date data. Each DBMS has its own variation of data types. Some basic constraints for data are NOT NULL, DEFAULT values, or UNIQUE.

Inserting data is a critical process regarding the accountability of the database. Data insertion must follow quality-designed insertion procedures to avoid erroneous and incomplete resulting information. When inserting into a table, the number of rows is increased by one. The SQL INSERT command inserts a single row of data into a table. When inserting data into a table, all the constraints that are specified during data creation must be applied.

When the data of the database are deleted or updated (modified), the database is changing. Careful steps must be taken when these activities are invoked to minimize the risk of losing valuable data. When deleting from a table, the number of rows is reduced by one or many. To delete data from a database, the DELETE SQL command is used, which may delete one or many rows or even the entire set of rows from a table.

Updating or modifying the data of a database means that certain fields of a row or rows will change. When updating a table, the number of rows remains the same. Only the data values inside the fields of the rows may change. The UPDATE SQL command is used to modify data. The integrity constraints CASCADE/RESTRICT/SET NULL/SET DEFAULT are applied to foreign keys and are included in the CREATE TABLE SQL command.

The role of these constraints is to ensure referential integrity; that means that the relationships between tables remain consistent; data and foreign keys must be consistent with the data in primary keys. Since by definition a foreign key is referenced to a primary key in another table, this key must exist in the primary table.

The ALTER TABLE SQL command is to modify the structure column definitions of a table. In general, CREATE TABLE and ALTER TABLE privileges in a database management system must be carefully granted. Usually, only database administrators have the right to make some modifications to the database.

Key Terms

SQL	CREATE DATABASE	CREATE TABLE
Data Types	Data Constraints	E−R Diagram
Primary Key Constraint	Foreign Key Constraint	Numeric Data Types
Date Data Types	Text Data Types	NOT NULL Constraint
DEFAULT Constraint	UNIQUE Constraint	CHECK Constraint
INSERT INTO	VALUES Clause	DELETE FROM
Conditional Operators	Logical Operators	DROP TABLE
UPDATE TABLE	ON DELETE	ON UPDATE
CASCADE	RESTRICT	SET NULL
SET DEFAULT	Referential Integrity	Orphan Child
ALTER TABLE		

Review Questions

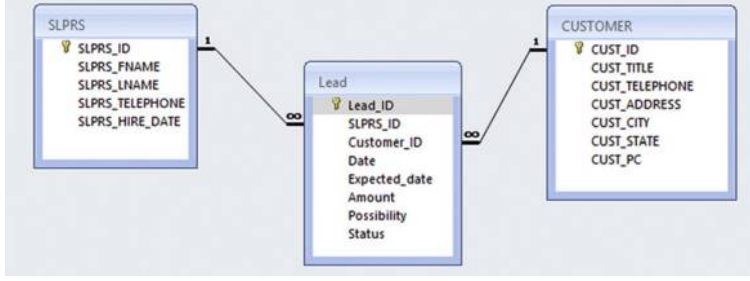
(1) Describe the transformation process from an ERM to database implementation.
(2) Discuss the phrase. "A database at implementation phase consists of **only** 1: M relationships."
(3) Should tables without any relationship to other tables should remain in the database. If not, what should be done?
(4) Describe the data types and constraints that are used in your working DBMS.
(5) Using desk research, identify some good-quality practices that will prevent erroneous data entry.
(6) Describe the database ON DELETE and ON UPDATE constraints to ensure referential integrity.

Problems and Exercises

(1) **Choose one of the cases of Chapter 3. After you have implemented the Entity Relationship Model (ERM), complete the following:**
   a. Create the necessary tables using the appropriate constraints when it is appropriate.
   b. In table creation, apply the necessary ON DELETE and ON UPDATE constraints.
   c. Establish the relationships that are mapped in your ERM.
   d. Insert test data into the database. Make sure that referential integrity rules are applied between primary and foreign keys during data entry. That means that when trying to insert a value as a foreign key while this value does not exist in the primary key, it should not be allowable.
   e. Test the database ensuring that data are stored efficiently and all constraints that you have imposed are applied. Provide evidence of your test cases. Highlight any problems discovered during the test phase and take appropriate corrective actions in database design to eliminate any problems.

(2) **In Exercise 2.6, related to sales forecasting, the following business requirements were set. A-Oil & Chemical is a chemical company that plans to create a database to forecast sales.**
   - A salesperson is responsible for a lead to sales. Each lead consists of the responsible salesperson, the customer targeted, date of occurrence, projected date, projected sale amount, and possibility of the sale to occur.
   - Each Salesperson is specified by first name, last name, telephone, and date of hire.
   - Each customer is specified by title, address, and telephone.
   - Leads that became sales are marked as "s" for success. Leads that fail are marked as "F" for failure. Leads that do not have a final outcome yet are marked as "I" for idle.

The following diagram reflects the design for the database.



a. **Write the SQL commands to create the three tables. Notice the relationships between keys and enforce on delete cascade and on update cascade constraints. Use appropriate data types for the working DBMS environment.**
b. **Answer questions *i through xiii* using the following data from the LEAD table.**

Lead_ID	SLPRS_ID	Customer_ID	Date	Expected_date	Amount	Possibility	Status
1	137	1002	23-Jul-15	15-Jun-16	10000	70	S
2	439	1007	11-Oct-14	12-Dec-14	12000	50	F
3	281	1006	23-Jul-15	30-Jul-17	20000	80	I
4	385	1004	13-Dec-15	20-Dec-16	30000	50	I
5	237	1009	10-Oct-15	20-Dec-15	15000	60	S
6	612	1014	23-Mar-16	20-Dec-16	20000	50	I
7	301	1003	17-Dec-15	20-Jun-16	12000	90	F

i. Write an SQL command to insert a new row into the Lead table including all fields.
ii. Write an SQL command to insert a row into the Lead table omitting Date and Amount data fields. Would this be allowable if the Amount attribute is created as NOT NULL?
iii. What will happen if you insert a new record with Lead_ID = 7?
iv. Write an SQL command to delete the row with possibility = 70.
v. Write an SQL command to delete all the rows from the table with Status = 'I'.
vi. Write an SQL command to delete all rows from the table with dates in 2015.
vii. Write an SQL command to delete all rows from the table with dates in 2015 and status 'F'. How many rows will be deleted?
viii. Write an SQL command to delete all rows with possibility = 50. How many rows will be deleted?
ix. Write an SQL command to delete the rows with possibility = 30. What happens if no such rows exist in the table for deletion?
x. Write an SQL command to delete the rows with SLPRS_ID = 237 and Customer_ID not equal to 1009. How many rows will be deleted?
xi. Write an SQL command to delete all rows that have an expected date before 1/1/2016 or an Amount less than 25000. How many rows will be deleted?
xii. Write an SQL command to delete all rows from the table.
xiii. What will be the resulting LEAD table after the execution of the following SQL Command?

```
DELETE FROM LEAD WHERE CUSTOMER_ID <> 1004
AND AMOUNT BETWEEN 20000 AND 30000;
```

(3) **For the customer service database, assume the following data for the CUSTOMER table.**

CUST_ID	CUST_TITLE	CUST_TELEPHONE	CUST_ADDRESS	CUST_CITY	CUST_STATE	CUST_PC
1000	ACME	34552334	West end 34	Los Angeles	CA	55124
1001	ORAMIS	36554221	Bond Drive 22A	Los Angeles	CA	55124
1002	CCT	34229642	21 Av. 21	New York	NY	37221
1003	AdV-Game	36441276	29th str 85	New York	NY	37221
1004	Galveston Inc	38772972	Orange str 32	Houston	TX	77648
1005	Beaumont Ind	31345729	Alina Str 87	Houston	TX	77648
1006	ARMENIA	34881294	Andron Av. 33	San Diego	CA	55120
1007	AST Co	37438765	Monterey Rd 77	San Diego	CA	55120
1008	BBCo	38765439	North End 922	San Antonio	TX	77501
1009	Astir	39723455	Art Av. 230	San Antonio	TX	77501
1010	NortStar	37665238	Antin Rd. 334	Albany	NY	37300
1011	Amedia	32887345	Aston Str 22	Allbany	NY	37300
1012	RamAI	34881277	Rami str. 30	Los Angeles	CA	
1013	ChaosLTD	33476323	One str 33	San Diego	CA	55120
1014	Arion	36446866	Felow av 289	Houston	TX	
1015	GJhInc	39824561	Alamo Str 349	San Antonio	TX	77501
1016	haTag	37665396	35 av 120	New York	NY	
1017	INU Co	35447326	Neon str 102	Albany	NY	37300

a. Write an SQL command to update the CUST_TITLE to "AMI" for the customer with CUST_ID = 1008.
b. Write an SQL command to update all records that have postal code CUST_PC = '55120' to '57120'.
c. Write an SQL command to update the CUST_PC to '32377', for all records that have CUST_PC NULL.
d. Write an SQL command to update the CUST_PC to '77777', for all records that have CUST_PC NOT NULL.
e. Write an SQL command to update the CUST_STATE to "VA" from all records that have CUST_PC = '77648'.
f. Write an SQL command to update the CUST_TELEPHONE to 33333333 for all records that have a CUST_TITLE starting with the character 'A'.
g. Write an SQL command to update the CUST_TELEPHONE to 33333333 for all records that have a CUST_TITLE ending with the character 'r'.
h. Write an SQL command to update the CUST_TELEPHONE to 33333333 for all records that have a CUST_TELEPHONE with '4' as the second character, for example, 34658638.
i. Write an SQL command to update the CUST_ADDRESS to '57 Jump str.', for all records that have a CUST_CITY = 'Los Angeles' and CUST_STATE = 'CA'. How many records will be updated?
j. Write an SQL command to update the CUST_ADDRESS to '57 Jump str.', for all records that have a CUST_CITY = 'Los Angeles' or CUST_STATE = 'TX'. How many records will be updated?
k. What will be the resulting CUSTOMER table after the execution of the following SQL Command?
   UPDATE CUSTOMER SET CUST_TITLE = 'AI' WHERE
   CUST_STATE <> 'CA' AND CUST_CITY = 'New York'

(4) **For the Salesperson table (SLPRS) of the service database, assume the following data in conjunction with LEAD table data (exercise 3.2) and CUSTOMER table (exercise 3.3).**

SLPRS_ID	SLPRS_FNAME	SLPRS_LNAME	SLPRS_TELEPHONE	SLPRS_HIRE_DATE
120	Rio	Astros	38756499	1/27/2014
137	Ann	McArthour	34337669	1/5/2013
177	Jane	Streinz	392237692	5/6/2014
211	Peter	Santos	34221556	6/7/2014
237	Andrea	Amelah	36442745	8/8/2013
281	Sam	Koori	38771990	7/28/2014
301	Johm	Jameson	37562341	10/9/2014
342	Kate	Arrow		12/10/2013
385	Fred	Feders	31228621	9/11/2014
415	Jasmin	Alorsa	334812312	2/12/2014
439	Ben	VanBerten		4/13/2013
473	Norma	Scott	35490432	10/14/2014
507	Maria	Salonikia	387665898	5/15/2014
522	Walter	Heizenberg	35439771	3/16/2013
547	Georgia	Kalamari	37847427	10/17/2014
612	Kim	Astor		1/18/2014
638	Chin	Young	32177663	7/19/2013
654	Jill	Adams	39881009	1/29/2014
691	Noo	Yamachi	37531951	2/20/2014
700	Dorian	Lee	36554772	4/30/2013
707	Faisal	AlGamdhi	33285289	8/21/2014
750	Pepito	Gonzales	37654991	10/22/2014
771	Nick	Manner		11/1/2013
831	Amanda	Ronaldo	32775985	10/23/2014
882	Grace	Ramos	33224593	9/2/2014

a. **Assuming the following ON DELETE AND ON UPDATE constraints on table LEAD:**

FOREIGN KEY	CONSTRAINT	
SLPRS_ID	ON DELETE	CASCADE
	ON UPDATE	CASCADE
CUST_ID	ON DELETE	RESTRICT
	ON UPDATE	CASCADE

How will the Customer service database tables be affected when SLPRS_ID = 385 is deleted from the SLPRS table and CUST_ID = 1009 is deleted from the CUSTOMER table.

b. **Assuming the following ON DELETE AND ON UPDATE constraints on table LEAD:**

FOREIGN KEY	CONSTRAINT	
SLPRS_ID	ON DELETE	NO CONSTRAINT
	ON UPDATE	CASCADE
CUST_ID	ON DELETE	CASCADE
	ON UPDATE	CASCADE

How will the Customer service database tables be affected when SLPRS_ID = 385 is deleted from the SLPRS table and CUST_ID = 1009 is deleted from the CUSTOMER table?

c. **Assuming the following ON DELETE AND ON UPDATE constraints on table LEAD:**

FOREIGN KEY	CONSTRAINT	
SLPRS_ID	ON DELETE	RESTRICT
	ON UPDATE	CASCADE
CUST_ID	ON DELETE	RESTRICT
	ON UPDATE	CASCADE

i. How will the Customer service database tables be affected when SLPRS_ID = 385 is deleted from the SLPRS table and CUST_ID = 1009 is deleted from the CUSTOMER table?
ii. Assuming that SLPRS_ID = 385 MUST deleted from the SLPRS table and CUST_ID = 1009 MUST deleted from the CUSTOMER table. Taking under consideration the RESTRICT constraint on the foreign keys, what is the correct order of DELETE commands to achieve that?
iii. How will the Customer service database tables be affected when SLPRS_ID = 385 is updated to 999 in the SLPRS table and CUST_ID = 1009 is updated to 7777 in the CUSTOMER table?