- Table names for all of logical entities. These table names would be the same as the actual table names in the database.
- Physical data types and sizes for all of the attributes. In the physical model, we will supply the data types of the entity attributes, such as `varchar`, `int`, `float`, and so on, along with the data type size. Because each database has its own unique set of physical data types, the physical model is dependent upon the type of database being used.
- Mapping tables or cross tables are used for storing many-to-many relationships within entities. A logical data model might not include mapping tables, but might simply depict such many-to-many relationship between entity A and B using m:n notation. But in a database, such relations need to be stored in another table, which we can refer to as cross or mapping table. Such tables are depicted in the physical data model (we will see one such example soon).
- Primary keys, foreign keys, indexes, and so on.

A primary key is an attribute which uniquely identifies a record in a table. If a primary key contains two or more attributes, it is called as a Composite Key.

A physical design represents the complete model of the database. It is the schema of the database being designed. The actual selection of the data types is dependent on the database system being used. We can do this either by creating or modifying the tables in the database or by using a modeling tool such as Microsoft Visio. Such tools are immensely useful for creating and managing data models, as one can generate database schema as well as create and share database diagrams based on different database providers.

While working on the physical model, we need to make sure that we use the right data types. Also, we need to be consistent with the data types. For example, all unique identifier columns (like the primary keys) should be of one data type. If we decide to support localized data, then we should use `nvarchar` instead of `varchar`. A proper selection of data types would ensure that our data model is scalable as well as flexible, in order to incorporate future changes.

Data Integrity

Referential integrity means that all of the primary and foreign keys in a database are valid, and no invalid links exist between the various tables that make up the system. Let's understand this with a simple example. Assume we have a `Customer` table, with a primary key of `CustomerID` and other related fields such as `First Name`, `Last name`, and so on.

Now we enter our first customer, and set the `CustomerID` to 1. Then we enter a second customer and set the new row's `CustomerID` to 2. Now if we update this second row, and change its `CustomerID` to 1, then the data integrity of our database system would be compromised, as we will have multiple customers with the same `CustomerID`.

To avoid such issues, we can use the Identity feature of SQL Server to auto-increment the primary key values. We will learn more about using identities later in this section.

Also, data integrity has to be maintained when using foreign keys. Foreign keys help in creating relationships between the two tables, where we create a reference by using the primary key of one table as a foreign key in the other. So if the row being referenced in the primary table is deleted, we need to make sure that we delete all references to that primary key from any related tables; otherwise the links would be left "hanging", or pointing to nothing.

We can maintain referential integrity by creating relationships in SQL Server. We will see an example of this later in this section.

Normalization

Normalization is a very important step towards creating a functional and efficient database model. Normalization is a process whereby we remove redundant data from tables and save valuable disk space by minimizing duplication of data. In this book we will not go into the basic process of normalization and the different normal forms, but will have a look at how under-normalization and over-normalization can cause issues and should be avoided.

To understand normalization in depth, refer to this book:

C. J. Date (1999), *An Introduction to Database Systems (8th edition)*. Addison-Wesley Longman. ISBN 0-321-19784-4.