The Data Hierarchy

Every organization needs to manage data about entities. An entity is a person, place, object, event or idea about which an organization stores data. Colleges maintain records regarding students, courses, faculty, equipment, and grades. Businesses need to keep track of customers, sales, employees, inventory and payments. A database management program is used to store, maintain and present data in an efficient and structured fashion. The elements of this structure are illustrated in Figure 1 and are described in the bulleted paragraphs below.

![Figure 1](#)

- A **field** is an attribute of an entity. You may think of a field as a category of data. There are 8 fields in the Customers table in Figure 1, including First, Last and Birth Date.

- A **record** is a collection of related fields, each pertaining to the same entity. The bottom row in the Customers table is the collection of fields for customer Scott Owen. There are 17 records in the Customers table.

- A **table** is a collection of related records, each having the same field structure. As you see in Figure 1, a table is a two-dimensional structure where the columns represent fields and the rows represent records. Tables are used to store data.

- A **database** is a collection of related tables and other objects, such as views, sequences, and indexes.

Keys and Relationships

In order to distinguish one record from any other record, each table should have a primary key defined. A primary key is a field (or a combination of fields) that uniquely identifies each record in a table. The CustID field can serve as the primary key in the Customers table since each record has a different value for CustID. Similarly, InvoiceNo serves as the primary key in the Customers table since each record has a different value for InvoiceNo. In a properly designed database, each table contains fields pertaining to a different entity. Since different fields are needed for customers than for sales, each entity’s fields are placed in a separate table and we end up with two tables: Customers and Sales. Although sales data is stored in a separate table, we need to be able to link each sale to the appropriate customer. You may have noticed that CustID also appears as a field in the Sales table. This allows us to link between customers and sales. When the primary key of one table is also included in another table for the purpose of linking related records, the field is called a foreign key. Thus, CustID is common to each table—it is the primary key of the Customers table and appears as foreign key in the Sales table so each sale can be linked to a customer.
If you scan the CustID field in the Sales table you’ll notice the value 11 appears twice, indicating that two sales have been made to customer 11, Kristen Reis. Thus, each customer can be associated with many sales. These two tables exhibit a **one-to-many** relationship, which is engineered by including the primary key of the one table as a foreign key in the many table. Figure 2 vividly illustrates this **one-to-many** relationship.

With this design, each customer’s fields are stored in a record in the Customers table and each sale’s fields are stored in a record in the Sales table. Each record in the Sales table also includes the CustID of the customer the sale was made to. When a customer has multiple sales, there will be multiple records in the Sales table with their CustID. Thus, each customer can be related to many sales, but each sale is made to one customer.

**Data Redundancy**

The preceding discussion focused on a properly designed database, where each entity is placed in a separate table and where a common field is available to join related records between the tables. We now explore the perils of an improper database design.

Take a few moments to inspect Figure 3. Notice that all of the fields now appear in a **single** table. Although this may seem simpler, it turns out to be a terrible idea.

Look at the third and fourth rows in Figure 3. These two rows reflect the sales made to Kristen Reis. But look more closely and you’ll notice that Kristen’s information (First, Last, Street, City, State, Zip and Birth Date) is stored twice because she has been the customer in two different sales. If she had been the customer in 50 sales, this design would require that we store her name, address and birth date 50 different times! Storing the same field values more than once (unnecessarily) is referred to as **data redundancy**.

Three problems are caused by data redundancy. The first is that storing values multiple times wastes space. Under a proper design (Figure 1), Kristen’s information is stored only once, in her record in the Customers table. The second problem is that when a field value changes, multiple occurrences need to be updated. For example, if Kristen moves, we’ll need to change the values for her Street, City, State and Zip in multiple records. The third problem occurs if we forget to change the values in any of the records. The database would then have inconsistent data.

**Summary**

With a proper design (Figures 1 and 2), each customer’s field values are stored only once, in a uniquely identifiable record in the Customers table. When a sale occurs, a record is created in the Sales table that includes a CustID that allows the sale to be linked to a specific customer. When a customer has many sales, multiple records will exist in the Sales table, each with that customer’s unique CustID. This design avoids data redundancy and the three problems associated with unnecessarily storing the same field values multiple times.