



1

Basic Relational Database Design

SQL is an acronym for Structured Query Language. The SQL language was designed to work with relational databases and is the universal language for accessing and manipulating the information stored within relational databases. Before we begin understanding how to code using the SQL language, we first need to understand the structure of relational databases to help us use the SQL language to its full potential.

CHAPTER OBJECTIVES

This chapter will introduce you to designing the logical structure of a database by using entity-relationship modeling and normalization guidelines. The objectives for this chapter are as follows:

- Understand how relational databases relate to business.
- Understand basics of entity-relationship design.
- Understand primary and foreign key purpose.
- Understand normalization guidelines.

Purpose of Databases

We create databases to store information related to business needs. Database structure allows the storing of business information in an organized format to help organizations conduct their daily business. The list below provides a few examples of business need and the possible types of information required to run the business.

| Business Need | Possible Database Name | Information |
|--|------------------------|---|
| A college needs a database to manage the students, teachers, programs of study, and classes. | ACADEMICS | Students, Teachers, Classes, Matriculations |
| A company needs to manage its employees, customers, and sales. | RETAIL | Employees, Customers, Sales, Purchase Orders, Inventory |
| An automobile dealer needs to manage its inventory. | INVENTORY | Automobiles, Services, Manufacturing Plant |
| A company needs to manage its projects for each department. | FINANCE_DEPT | Projects, Employees, Resources, Expenses |

Entities

Entities in database design consist of objects or things. For example, students, teachers, and classes are all entities. For a company, employees, departments, and addresses are all entities. When beginning the design of a database, a list of entities should be created. By creating a list of entities prior to designing the database, the designer will be better prepared for understanding the needs of the database.

Throughout this book we will be working with the Human Resources database. The core entities we've identified for this database are as follows:

1. Employees
2. Departments
3. Addresses
4. Employees Dependents

Primary Keys

A primary key is a column that contains a unique identifier for the data contained in the row. A primary key must always be unique and two identical primary keys cannot exist in the same row. Primary keys MUST NEVER change. They are the one unique identifier on a row.

Intelligent Primary Keys

When choosing a valid primary key the Database Designer should avoid using "intelligent" primary keys. Intelligent primary keys are keys that provide information. In the example below the EmployeeID

is prefixed with two letters representing the region that the employee works at for the company. For example, “NE” represents the northeast region while “SW” represents the southwest region.

Employees

| EmployeeID | FirstName | LastName | DOB | Title |
|------------|------------|----------|------------|------------------------|
| NE1001 | Thomas | Smith | 09/19/1980 | Business Analyst |
| NE1002 | Brenda | Jackson | 08/17/1992 | Data Scientist |
| SE1003 | Sally | Monarca | 11/27/1975 | Accounting Clerk |
| SW1004 | Angelo | Romano | 02/22/1981 | Desktop Specialist |
| NW1005 | Antoinette | Milardo | 09/15/1971 | Project Manager |
| SE1006 | Carla | Bombaci | 05/17/1994 | Director of Accounting |

Referring to the table above, what happens if Thomas Smith is transferred to the southwest region? “NE” would need to be replaced with “SW” to accurately identify region of employment. However, this would violate primary key rules. Primary keys MUST NEVER be changed.

Surrogate Primary Key

Surrogate primary keys are keys that have no meaning (intelligence) other than to uniquely identify a row. The database has the ability to assign surrogate primary keys in sequence that will be unique to each row in the table. There will NEVER be a duplicate. The database ensures that.

In the modified Employees table below you can see the new EmployeeID is made up of surrogate keys created by the database. Now the region is no longer part of the primary key but just another column in the table. So Thomas can transfer to the southwest region and his employee id will stay the same. Instead the region will be changed accordingly. This is ok since it is not part of the primary key.

Table 1

| EmployeeID | FirstName | LastName | DOB | Title | Region |
|------------|------------|----------|------------|------------------------|--------|
| 1001 | Thomas | Smith | 09/19/1980 | Business Analyst | NE |
| 1002 | Brenda | Jackson | 08/17/1992 | Data Scientist | NE |
| 1003 | Sally | Monarca | 11/27/1975 | Accounting Clerk | SE |
| 1004 | Angelo | Romano | 02/22/1981 | Desktop Specialist | SW |
| 1005 | Antoinette | Milardo | 09/15/1971 | Project Manager | NW |
| 1006 | Carla | Bombaci | 05/17/1994 | Director of Accounting | SE |

Foreign Keys

Foreign keys are used to create links between tables. For example, the Class table shown below has a *Class ID* column as its primary key (background color orange). The Teachers table also contains a column entitled *Class ID* but it serves as a foreign key rather than a primary key. There is an established link between the two *Class ID* columns: The *Class ID* column from Teachers connects to the *Class ID* in Classes. So, we know which classes Thomas Smith teaches because we can connect the foreign key of *Class ID* in Teachers to the primary key of *Class ID* in Classes. So, Thomas Smith teaches English 101.

Teachers

| Teacher ID | Class ID | First Name | Middle Name | Last Name |
|------------|----------|------------|-------------|-----------|
| 1 | 8 | Thomas | L | Smith |
| 2 | 10 | Robert | A | Bevans |
| 3 | 5 | Sally | B | Viola |
| 5 | 22 | Tammy | | Jackson |

Classes

| Class ID | Class Name | Department |
|----------|------------------|------------------|
| 8 | Java | Computer Science |
| 10 | Intro to MIS | Computer Science |
| 5 | Calculus I | Math |
| 15 | Databases Design | Computer Science |
| 19 | Algebra | Math |
| 21 | Statistics | Math |
| 22 | English 101 | English |
| 24 | English 102 | English |

Database Normalization

Database normalization is the standard structure for relational databases. The objective of normalization is to design the database in a way that reduces data redundancy, minimizes the need of database redesign when extending the database, and creating a properly functioning database. To comply with database normalization, databases must meet three sets of rules to be considered normalized.

First Normal Form

First normal form dictates that each cell be atomic, meaning that one value and only one value can be stored in a cell. First normal form also requires that repeating groups not exist. Table 1a is an example of proper first normal form.

Table 1a

| Employee ID | First Name | Middle Name | Last Name | Dependents |
|-------------|------------|-------------|-----------|------------|
| 1 | Thomas | L | Smith | Bobby |
| 2 | Robert | A | Bevans | Margaret |
| 3 | Sally | B | Viola | Paul |

Table 2a, as shown below, violates the rule of first normal form because Thomas Smith and Sally Viola have multiple dependents under the same cell, “Dependents”. *See cells in yellow.*

Table 2a

| Employee ID | First Name | Middle Name | Last Name | Dependents |
|-------------|------------|-------------|-----------|----------------------------------|
| 1 | Thomas | L | Smith | Bobby Mark Valerie Tina |
| 2 | Robert | A | Bevans | Margaret |
| 3 | Sally | B | Viola | Paul Cheryl |

One way to resolve lack of autonomy in Table 2a is to add more columns as seen in Table 3a.

Table 3a

| Employee ID | First Name | Middle Name | Last Name | Dependent 1 | Dependent 2 | Dependent 3 | Dependent 4 |
|-------------|------------|-------------|-----------|-------------|-------------|-------------|-------------|
| 1 | Thomas | L | Smith | Bobby | Mark | Valerie | Tina |
| 2 | Robert | A | Bevans | Margaret | | | |
| 3 | Sally | B | Viola | Paul | Cheryl | | |

While Table 3a no longer has more than one value in a cell, it still violates first normal form because it has repeating groups. Repeating groups should be avoided for the two major reasons listed below:

1. Since you don't know how many dependents employees may have during their career, you may be required to change the table structure endlessly to accommodate all the possible dependents by adding more columns.
2. For employees who either have no dependents or less than the maximum number of columns that allow for dependents, database space is wasted because the structure exists for them but they're not utilizing it. *See cells in yellow (Table 3a).*

We will investigate how to solve these complications as we continue with normal forms.

To summarize, to adhere to the requirements of first normal form the following criteria must be met:

1. Table cells must be autonomous.
2. Avoid repeating groups.
3. Create a separate table for each set of data; in our example, dependents.
4. Each set of related data must have a primary key.

Second Normal Form

Second normal form requires a table to be in first normal form and all nonprimary key columns relate to the entire primary key. In Table 1b, the dependents' names along with the *Employee ID* make it unique (background color yellow). This is considered a composite key because the unique identifier is requiring more than one column. However, since an employee's first name, middle name, and last name have nothing to do with the dependent name, it is in violation of second normal form.

Table 1b

| Employee ID | First Name | Middle Name | Last Name | Dependents | Department |
|-------------|------------|-------------|-----------|------------|------------------|
| 1 | Thomas | L | Smith | Bobby | Warehouse |
| 1 | Thomas | L | Smith | Mark | Warehouse |
| 1 | Thomas | L | Smith | Valerie | Warehouse |
| 1 | Thomas | L | Smith | Tina | Warehouse |
| 2 | Robert | A | Bevans | Margaret | Sales |
| 3 | Sally | B | Viola | Paul | Sales |
| 3 | Sally | B | Viola | Cheryl | Customer Service |


The solution is seen below where two tables exist, one for employees and the other for employees' dependents. Each table supports first normal form.

Notice the relationship between the Employees table and the Employees Dependents table using *Employee ID*. This is a one-to-many relationship. Employees can have more than one dependent. The *Employee ID* is the primary key and is unique. In the Employees Dependents table, the *Employee ID* is the foreign key and can be repeated. It links the appropriate dependents to the employee.

► **Note:** Notice, also, that the Employees Dependents table has its own primary key.

Employee

| Employee ID | First Name | Middle Name | Last Name | Department |
|-------------|------------|-------------|-----------|------------------|
| 1 | Thomas | L | Smith | Warehouse |
| 2 | Robert | A | Bevans | Sales |
| 3 | Sally | B | Viola | Customer Service |

Employee Dependents


| Dependent ID | Employee ID | Dependent Name |
|--------------|-------------|----------------|
| 1 | 1 | Bobby |
| 2 | 1 | Mark |
| 3 | 1 | Valerie |
| 4 | 1 | Tina |
| 5 | 2 | Margaret |
| 6 | 3 | Paul |
| 7 | 3 | Cheryl |

In our example above, the relationship between Employee and Dependents is one to many. The employee can have one or more dependents.

In summary, second normal form requires the following criteria to be met:

1. Tables must be in first normal form.
2. All nonprimary key columns relate to the entire primary key.

Third Normal Form

Third normal form requires that data not dependent on the primary key be removed. In the Employees table shown below, all the data is dependent on the *EmployeeID*, except department. Department should have its own table, where data such as location, phone, and manager in charge can be stored.

Employees

| Employee ID | First Name | Middle Name | Last Name | Department |
|-------------|------------|-------------|-----------|------------------|
| 1 | Thomas | L | Smith | Warehouse |
| 2 | Robert | A | Bevans | Sales |
| 3 | Sally | B | Viola | Customer Service |

When conforming to third normal form we need to separate Employees and Departments into separate tables as shown below.

Employees

| Employee ID | First Name | Middle Name | Last Name |
|-------------|------------|-------------|-----------|
| 1 | Thomas | L | Smith |
| 2 | Robert | A | Bevans |
| 3 | Sally | B | Viola |
| 4 | Susan | N | Stevens |
| 5 | Mark | C | Jackson |

Departments

| Department ID | Department Name | Location |
|---------------|------------------|------------------------------|
| 1 | Warehouse | 21 Alexander Ave, Office 205 |
| 2 | Sales | 401 Greenwich Ave |
| 3 | Customer Service | 401 Greenwich Ave, Room 100 |
| 4 | Repair | 401 Greenwich Ave, Room 105 |

Next, we need create a relationship between the two tables. To do that we must determine what type of relationship exists by asking the following questions:

1. **Can an employee belong to more than one department?** Yes, an employee may work for many different departments during a long career with the company.
2. **Can a department contain more than one employee?** That's simple enough to answer. The answer is yes.

So, given that an employee can work for more than one department and a department can have more than one employee, we have a many-to-many relationship. Anytime there is a many-to-many relationships we **MUST** add another table called a **BRIDGE** table that contains the primary keys of both tables we are building a relationship for. These primary keys serve as foreign keys, individually, that link back to the Employees and Departments tables.

Foreign keys can repeat themselves since they relate back to the primary key of a table. However, two or more foreign keys together can make up a primary key because together they are unique. In such a situation we call this a composite primary key.

The EmployeesDepartments table below is our **bridge** table that will be used to connect the Employees and Departments tables.

EmployeesDepartments

| Employee ID | Department ID |
|-------------|---------------|
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |
| 5 | 3 |
| 5 | 4 |
| 4 | 4 |

Figure 1 below is a diagram representing the relationship between Employees and Departments.

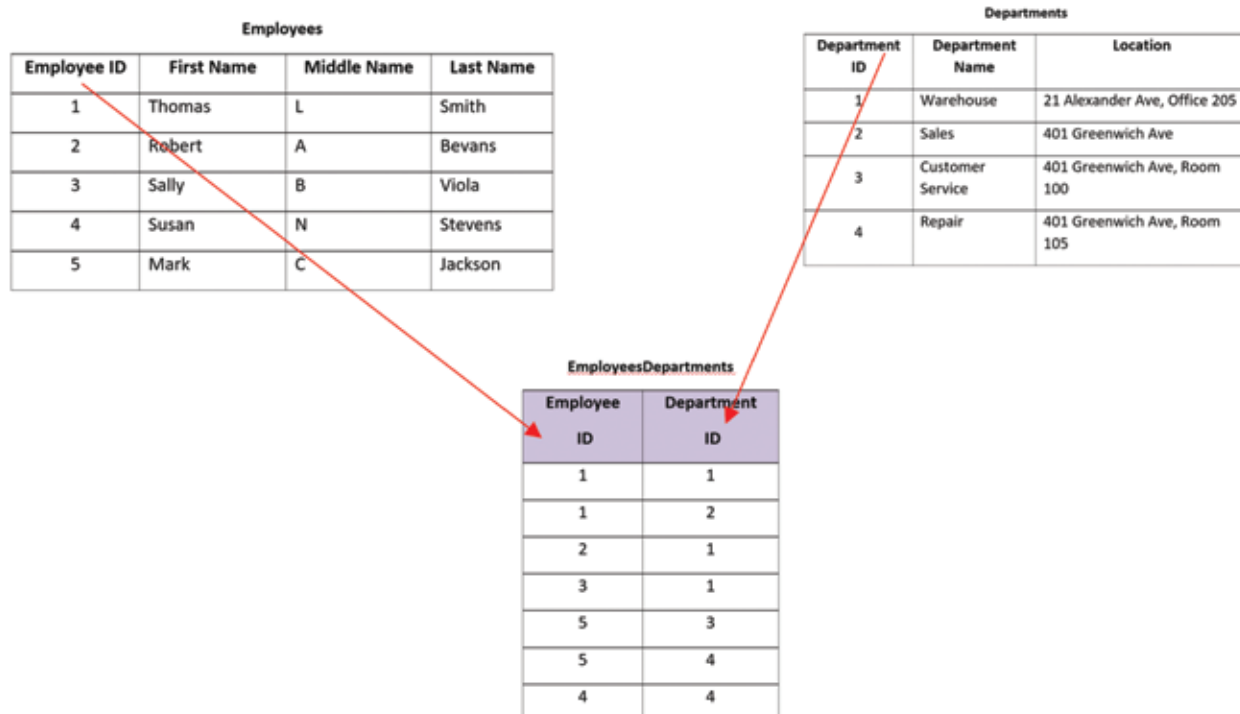


Figure 1

Now what happens if Thomas Smith decides that he doesn't like working for sales and now wants to go back to his old job at the ware house? The bridge table would have a duplicate composite primary key as shown below highlighted in yellow.

| Employee ID | Department ID |
|-------------|---------------|
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |
| 5 | 3 |
| 5 | 4 |
| 4 | 4 |
| 1 | 1 |

The result would be a violation of the primary key rules. To prevent this, it is common to add two more columns, “Effective Date” and “Term Date.” The effective date determines the date the employee started with the department and the term date determines when the employee ended employment with the department. Let’s take a look again at the table, only this time, we added effective date and term date.

EmployeesDependents

| Employee ID | Department ID | Effective Date | Term Date |
|-------------|---------------|----------------|------------|
| 1 | 1 | 01/05/2012 | 09/15/2016 |
| 1 | 2 | 09/16/2016 | 03/18/2017 |
| 2 | 1 | 08/07/2014 | |
| 3 | 1 | 12/17/2014 | |
| 5 | 3 | 01/02/2015 | 5/17/2015 |
| 5 | 4 | 07/08/2015 | |
| 4 | 4 | 09/16/2015 | |
| 1 | 1 | 03/19/2017 | |

Now with the addition with effective date and term date we can uniquely identify a row. To do so we include effective date as part of the composite key. We can see that employee 1 worked for department 1 starting on 01/05/2012 and ended on 09/15/2016. Then he started working for department 2 on 09/16/2016. Because the department is different, both the employee id and department id make it unique. However, if he works for department 1 again, the uniqueness is gone. But effective date is added and now the three columns, employee id, department id, effective date work together to form a composite primary key.

So, we can understand the following from viewing the EmployeesDepartments table.

In summary, it is important to remember when there is a many to many relationship a bridge table is required with foreign keys from the primary tables to create a unique composite key. Sometimes the composite key consisting of foreign keys alone is not sufficient to be a primary key because it may not be unique. In this case, another column is added to make it unique. For our example, effective date served this purpose.

Summary

This chapter introduced you to the how relational databases serve business organization and function by organizing, storing, and processing important data.

In addition, relational database design using normalization methods were covered. Database normalization includes the need for primary keys, foreign keys, and composite keys, which are used to uniquely identify rows of data and relate information.

The three normal forms were introduced and the chapter explained the necessity of the relational normalization concepts for proper database structure.

Exercises

1. Normalize the following to Third Normal Form

A hospital needs to track doctors and patients. There is a need to track what patients, doctors have seen and what doctors, patients have seen. Use Word to create tables that contain the following data entities and data elements.

Entity 1 (Doctor name, type of doctor, office location, date hired)

Entity 2 (Patient name, address, phone, date of service, type of service)

Hint: A doctor can have many patients. Patients can have many doctors. So, a many-to-many relationship exists.