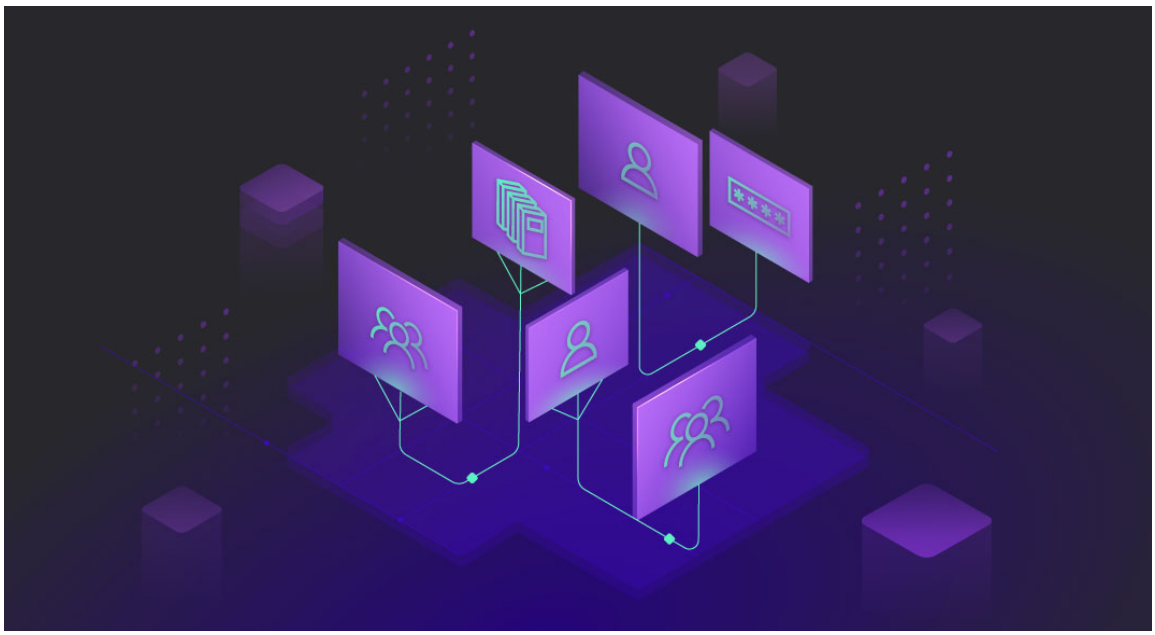
devart

Home › How To › Relationships in SQL - Complete Guide With Examples

How To SQL Server Tools

Relationships in SQL – Complete Guide With Examples

By **dbForge Team** October 26, 2021 71794 0



In this article, we are about to take a close look at the SQL Server relations, including one-to-one, many-to-many, and many-to-one relations, and get into details of creating those with the help of dbForge Studio for SQL Server.

Well-established relations in a database allow for clear table structures and help cut redundant data to a minimum. Building efficient relations in a database is extremely important— it helps enforce referential integrity which in its turn contributes to database normalization.

There are five types of relations in the databases: one-to-one, one-to-many, many-to-one, many-to-many, and self-referencing relationships. So, what's the difference between these database relationship types? In the article, we'll examine each type separately and provide a working example of their usage.

CONTENTS

What is a relation in a database?

Let's start with some basic terminology.

The term ***relation*** is sometimes used to refer to a table in a relational database. However, it is more often used to describe the relationships that exist between the tables in a relational database.

A ***relationship*** between two database tables presupposes that one of them has a foreign key that references the primary key of another table.

Database entity—strictly speaking — is a person, place, thing, object, or any item about which data is stored in the database. However, the term is usually used to refer to the database table as tables are, in fact, the physical implementation of entities.

An ***entity-relationship diagram***, also known as ERD, ER diagram, or ER model, comprises a graphical representation of how entities relate to each other within a database.
ER models are widely used in database design as they are fairly abstract and are easy to view and analyze.

Let's have a look at the entity-relationship diagram of the BicycleStore database with the help of the Database Diagram tool that comes with dbForge Studio for SQL Server.

Types of relationships in a database

There are 3 main types of relationship in a database:

- many-to-many.

However, you may also encounter references to a many-to-one relationship which, in fact, is a special case of a one-to-many relationship and self-referencing relationship which occurs when only one table is involved.

Let's examine each relationship type in detail and consider the peculiarities of creating relationships in SQL.

One-to-many relationship

Let's start with a one-to-many relationship as it is the most commonly used type. So, what is one-to-many relationship in SQL? A one-to-many relationship occurs when one record in table 1 is related to one or more records in table 2. However, one record in table 2 cannot be related to more than one record in table 1. We can come up with hundreds of examples of such relations: pages and the book they belong to, pupils and their class, orders and the customer who placed them, etc.

How to join tables with one-to-many relationship in SQL? INNER JOINs are considered to be the most effective way to combine data from two tables that have one-to-many relationship. Let's query two SQL tables having one-to-many relationship.

```
SELECT
  *
FROM country c
INNER JOIN city c1
  ON c.country_id = c1.country_id
```

1. Create two tables (table 1 and table 2) with their own primary keys.
2. Add a foreign key on a column in table 1 based on the primary key of table 2. This will mean that table 1 can have one or more records related to a single record in table 2.

Step 1

```
CREATE TABLE dbo.city (
  city_id int IDENTITY,
  city varchar(50) NOT NULL,
  country_id int NOT NULL,
  CONSTRAINT PK_city PRIMARY KEY CLUSTERED (city_id)
)
ON [PRIMARY]
GO


CREATE TABLE dbo.country (
  country_id int IDENTITY,
  country varchar(50) NOT NULL,
  CONSTRAINT PK_country PRIMARY KEY CLUSTERED (country_id)
)
ON [PRIMARY]
GO
```

Step 2

```
ALTER TABLE dbo.city WITH NOCHECK
  ADD FOREIGN KEY (country_id) REFERENCES dbo.country (country_id)
GO
```

One-to-one relationship

A one-to-one relationship in a database occurs when each row in table 1 has only one related row in table 2. For example, a department may have only one head manager, a husband — only one wife, an employee — one company car, etc.

One-to-one relationship example in SQL:

define a simple primary foreign key relationship between them, and set the foreign key column to be unique.

```
CREATE TABLE Employee (
  ID int PRIMARY KEY,
  Name VARCHAR(50)
);

CREATE TABLE Salary (
  EmployeeID int UNIQUE NOT NULL,
  SalaryAmount int
);

ALTER TABLE Salary
ADD CONSTRAINT FK_Salary_Employee FOREIGN KEY (EmployeeID)
REFERENCES Employee (ID);
```

Many-to-many relationship

A many-to-many relationship occurs when multiple records in one table are related to multiple records in another table. For example, products and suppliers: one supplier may deliver one or many products and at the same time, the company may order one product from one or many suppliers.

A many-to-many relationship example in SQL:

The relationship between the Product entity and Order entity is many-to-many, as one product may be in many orders and many orders may contain the same product.

Example of creating many-to-many relation in SQL

Relational databases don't support direct many-to-many relationships between two tables. Then, how to implement many-to-many relationships in SQL? To create a many-to-many relationship in a database, you'll need to create a third table to connect the other two. This new table (also known as a *linking*, *joining*, *bridging*, or *junction* table) will contain the primary key columns of the two tables you want to relate and will serve as an intermediate table between them.

Let's consider the following example of how to create many-to-many relationship in SQL. Suppose, we want to establish a many-to-many relationship between two tables: *films* and *category*. First, we create the two tables.

```
 ,director VARCHAR(50)
 ,year_released DATETIME
);


CREATE TABLE category (
  category_id INT PRIMARY KEY
 ,name VARCHAR(50)
);
```

Next, we create a junction table *film_category* that will map these two tables together by referencing the primary keys of both tables.

```
CREATE TABLE film_category (
  film_id INT
 ,category_id INT
 ,CONSTRAINT film_cat_pk PRIMARY KEY (film_id, category_id)
 ,CONSTRAINT FK_film
  FOREIGN KEY (film_id) REFERENCES films (film_id)
 ,CONSTRAINT FK_category
  FOREIGN KEY (category_id) REFERENCES category (category_id)
);
```

A many-to-many relationship between the films and category tables has been successfully established.

Many-to-one relationship

Many experts don't separate a many-to-one relationship as a class of its own as there is not much difference between one-to-many and many-to-one relationships. It's just a matter of focus. For example, if one school class can consist of several pupils then, class to pupil is a one-to-many relationship (one class consists of many pupils), while pupil to class relationship is many-to-one (many pupils study in one class).

Example of many-to-one relationship in SQL:

Self-referencing relationships

A self-referencing relationship (also known as a recursive relationship) in a database occurs when a column in a table relates to another column in the same table. In such a relationship, only one table is involved. For example, the *Staff* table contains information about company employees and their managers, however, managers themselves belong to staff too.

A self-referencing relationship example in SQL:

How to create a self-referencing relationship in a SQL Server database

To establish a self-referencing relationship on a table, we create a table, define a primary key, and then add a foreign key referencing that primary key.

```
CREATE TABLE Sales.Staff (
  StaffId int IDENTITY,
  FirstName varchar(50) NOT NULL,
  LastName varchar(50) NOT NULL,
  Email varchar(255) NOT NULL,
  Phone varchar(25) NULL,
  Active tinyint NOT NULL,
  StoreId int NOT NULL,
  ManagerId int NULL,
  PRIMARY KEY CLUSTERED (StaffId),
  UNIQUE (Email)
)
ON [PRIMARY]
GO
```

To learn more about establishing relationships in a database, please refer to our SQL Database Design Basics with Examples article.

Create table relationships in SQL Server using SQL Designer

The methods to define relationships in a SQL Server database described above are not so easy and straightforward. They involve a bit of coding and demand a certain level of SQL expertise. Is there an easier way to create relationships? Let's have a look at the method to implement relations visually—using the SQL Designer that comes with dbForge Studio for SQL Server.

Database Designer is a smart data modeling utility that allows you to visually build new databases and quickly analyze the structures of the existing ones. Thus, with dbForge Studio for SQL Server, you can effortlessly design your databases at both logical and physical levels and benefit from these capabilities:

- Drag-and-drop database objects from Database Explorer directly to a diagram
- Create and edit database objects on a diagram
- Group diagram objects in containers for better visibility
- Establish relations between the entities
- Monitor logical relationships between tables
- Reverse-engineer a database
- Export database diagrams as images
- Print out large SQL database diagrams
- Add notes, stamps, and images to a diagram

development from design to maintenance. dbForge Studio for SQL Server allows visualizing and tracking relations between tables in a few clicks. With Virtual Relations Manager you can create and edit virtual relations between tables and then convert them into foreign keys.

Conclusion

The article discusses the different types of entity relationships in a database: one-to-one, one-to-many, many-to-one, many-to-many, and self-referencing relationships. Managing database relations with bare hands can be a mind-blowing task. That's where dbForge Studio for SQL Server hits the stage delivering the mighty Database Diagram tool that effectively handles even the most complex database relationships.

Download a free 30-day trial of dbForge Studio for SQL Server right now and enjoy the Database Designer functionality along with many other robust features of the best SQL Server GUI tool you can find!