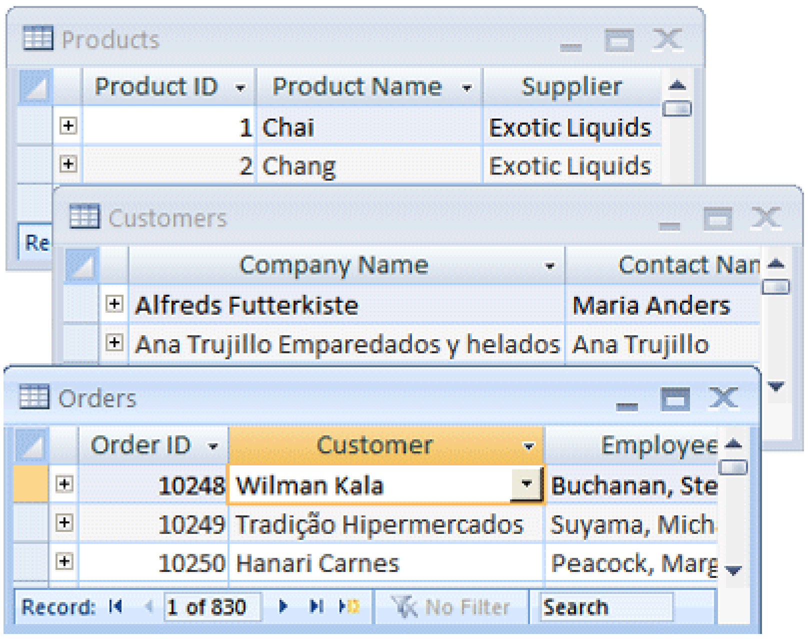
Some database terms to know

Microsoft Office Access 2007 organizes your information into **tables**: lists of rows and columns reminiscent of an accountant's pad or a Microsoft Office Excel 2007 worksheet. In a simple database, you might have only one table. For most databases you will need more than one. For example, you might have a table that stores information about products, another table that stores information about orders, and another table with information about customers.



Products

Product ID	Product Name	Supplier
1	Chai	Exotic Liquids
2	Chang	Exotic Liquids

Customers

Company Name	Contact Nan
Alfreds Futterkiste	Maria Anders
Ana Trujillo Emparedados y helados	Ana Trujillo

Orders

Order ID	Customer	Employee
10248	Wilman Kala	Buchanan, Ste
10249	Tradição Hipermercados	Suyama, Mich
10250	Hanari Carnes	Peacock, Marg

Each row is also called a **record**, and each column, is also called a **field**. A record is a meaningful and consistent way to combine information about something. A field is a single item of information — an item type that appears in every record. In the Products table, for instance, each row or record would hold information about one product. Each column or field holds some type of information about that product, such as its name or price.

What is good database design?

Certain principles guide the database design process. The first principle is that duplicate information (also called redundant data) is bad, because it wastes space and increases the likelihood of errors and inconsistencies. The second principle is that the correctness and completeness of information is important. If your database contains incorrect information, any reports that pull information from the database will also contain incorrect information. As a result, any decisions you make that are based on those reports will then be misinformed.

A good database design is, therefore, one that:

- Divides your information into subject-based tables to reduce redundant data.
- Provides Access with the information it requires to join the information in the tables together as needed.
- Helps support and ensure the accuracy and integrity of your information.
- Accommodates your data processing and reporting needs.

The design process

The design process consists of the following steps:

- **Determine the purpose of your database**

This helps prepare you for the remaining steps.

- **Find and organize the information required**

Gather all of the types of information you might want to record in the database, such as product name and order number.

- **Divide the information into tables**

Divide your information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.

- **Turn information items into columns**

Decide what information you want to store in each table. Each item becomes a field, and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.

- **Specify primary keys**

Choose each table’s primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID or student ID number.

- **Set up the table relationships**

Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.

- **Refine your design**

Analyze your design for errors. Create the tables and add a few records of sample data. See if you can get the results you want from your tables. Make adjustments to the design, as needed.

- **Apply the normalization rules**

Apply the data normalization rules to see if your tables are structured correctly. Make adjustments to the tables, as needed.

Determining the purpose of your database

It is a good idea to write down the purpose of the database on paper — its purpose, how you expect to use it, and who will use it. For a small database for a home based business, for example, you might write something simple like "The customer database keeps a list of customer information for the purpose of producing mailings and reports." If the database is more complex or is used by many people, as often occurs in a corporate setting, the purpose could easily be a paragraph or more and should include when and how each person will use the database. The idea is to have a well developed mission statement that can be referred to throughout the design process. Having such a statement helps you focus on your goals when you make decisions.

Finding and organizing the required information

To find and organize the information required, start with your existing information. For example, you might record purchase orders in a ledger or keep customer information on paper forms in a file cabinet. Gather those documents and list each type of information shown (for example, each box that you fill in on a form). If you don't have any existing forms, imagine instead that you have to design a form to record the customer information. What information would you put on the form? What fill-in boxes would you create? Identify and list each of these items. For example, suppose you currently keep the customer list on index cards. Examining these cards might show that each card holds a customers name, address, city, state, postal code and telephone number. Each of these items represents a potential column in a table.

As you prepare this list, don’t worry about getting it perfect at first. Instead, list each item that comes to mind. If someone else will be using the database, ask for their ideas, too. You can fine-tune the list later.

Next, consider the types of reports or mailings you might want to produce from the database. For instance, you might want a product sales report to show sales by region, or an inventory summary report that shows product inventory levels. You might also want to generate form letters to send to customers that announces a sale event or offers a premium. Design the report in your mind, and imagine what it would look like. What information would you place on the report? List each item. Do the same for the form letter and for any other report you anticipate creating.

Northwind Traders
Product Inventory

Product Id	Name	Quantity on Hand
1	Chai	36
2	Chang	17

Giving thought to the reports and mailings you might want to create helps you identify items you will need in your database. For example, suppose you give customers the opportunity to opt in to (or out of) periodic e-mail updates, and you want to print a listing of those who have opted in. To record that information, you add a “Send e-mail” column to the customer table. For each customer, you can set the field to Yes or No.

The requirement to send e-mail messages to customers suggests another item to record. Once you know that a customer wants to receive e-mail messages, you will also need to know the e-mail address to which to send them. Therefore you need to record an e-mail address for each customer.

It makes good sense to construct a prototype of each report or output listing and consider what items you will need to produce the report. For instance, when you examine a form letter, a few things might come to mind. If you want to include a proper salutation — for example, the "Mr.", "Mrs." or "Ms." string that starts a greeting, you will have to create a salutation item. Also, you might typically start a letter with “Dear Mr. Smith”, rather than “Dear. Mr. Sylvester Smith”. This suggests you would typically want to store the last name separate from the first name.

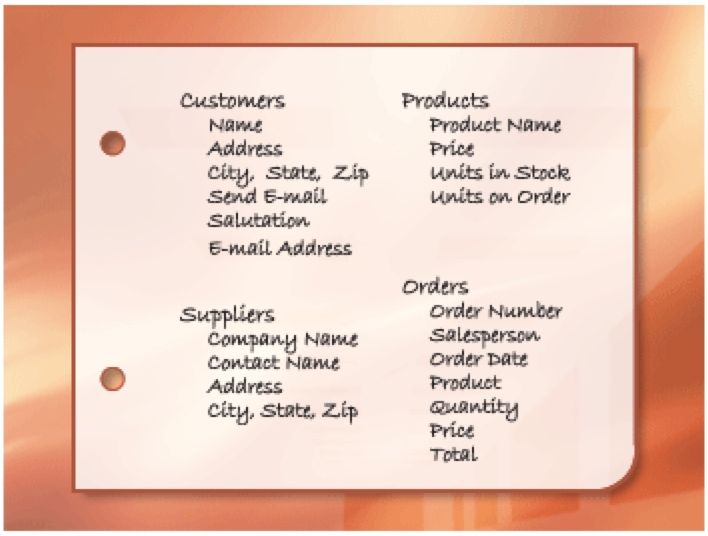
A key point to remember is that you should break each piece of information into its smallest useful parts. In the case of a name, to make the last name readily available, you will break the name into two parts — First Name and Last Name. To sort a report by last name, for example, it helps to have the customer's last name stored separately. In general, if you want to sort, search, calculate, or report based on an item of information, you should put that item in its own field.

Think about the questions you might want the database to answer. For instance, how many sales of your featured product did you close last month? Where do your best customers live? Who is the supplier for your best-selling product? Anticipating these questions helps you zero in on additional items to record.

After gathering this information, you are ready for the next step.

Dividing the information into tables

To divide the information into tables, choose the major entities, or subjects. For example, after finding and organizing information for a product sales database, the preliminary list might look like this:


Customers
- Name
- Address
- City, State, Zip
- Send E-mail
- Salutation
- E-mail Address

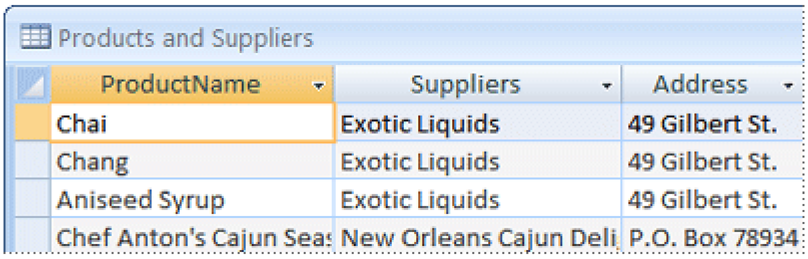
Suppliers
- Company Name
- Contact Name
- Address
- City, State, Zip

Products
- Product Name
- Price
- Units in Stock
- Units on Order

Orders
- Order Number
- Salesperson
- Order Date
- Product
- Quantity
- Price
- Total

The major entities shown here are the products, the suppliers, the customers, and the orders. Therefore, it makes sense to start out with these four tables: one for facts about products, one for facts about suppliers, one for facts about customers, and one for facts about orders. Although this doesn't complete the list, it is a good starting point. You can continue to refine this list until you have a design that works well.

When you first review the preliminary list of items, you might be tempted to place them all in a single table, instead of the four shown in the preceding illustration. You will learn here why that is a bad idea. Consider for a moment, the table shown here:


Products and Suppliers

ProductName	Suppliers	Address
Chai	Exotic Liquids	49 Gilbert St.
Chang	Exotic Liquids	49 Gilbert St.
Aniseed Syrup	Exotic Liquids	49 Gilbert St.
Chef Anton's Cajun Sea:	New Orleans Cajun Deli	P.O. Box 78934

In this case, each row contains information about both the product and its supplier. Because you can have many products from the same supplier, the supplier name and address information has to be repeated many times. This wastes disk space. Recording the supplier information only once in a separate Suppliers table, and then linking that table to the Products table, is a much better solution.

A second problem with this design comes about when you need to modify information about the supplier. For example, suppose you need to change a supplier's address. Because it appears in many places, you might accidentally change the address in one place but forget to change it in the others. Recording the supplier's address in only one place solves the problem.

When you design your database, always try to record each fact just once. If you find yourself repeating the same information in more than one place, such as the address for a particular supplier, place that information in a separate table.

Finally, suppose there is only one product supplied by Coho Winery, and you want to delete the product, but retain the supplier name and address information. How would you delete the product record without also losing the supplier information? You can't. Because each record contains facts about a product, as well as facts about a supplier, you cannot delete one without deleting the other. To keep these facts separate, you must split the one table into two: one table for product information, and another table for supplier information. Deleting a product record should delete only the facts about the product, not the facts about the supplier.

Once you have chosen the subject that is represented by a table, columns in that table should store facts only about the subject. For instance, the product table should store facts only about products. Because the supplier address is a fact about the supplier, and not a fact about the product, it belongs in the supplier table.

Turning information items into columns

To determine the columns in a table, decide what information you need to track about the subject recorded in the table. For example, for the Customers table, Name, Address, City-State-Zip, Send e-mail, Salutation and E-mail address comprise a good starting list of columns. Each record in the table contains the same set of columns, so you can store Name, Address, City-State-Zip, Send e-mail, Salutation and E-mail address information for each record. For example, the address column contains customers' addresses. Each record contains data about one customer, and the address field contains the address for that customer.

Once you have determined the initial set of columns for each table, you can further refine the columns. For example, it makes sense to store the customer name as two separate columns: first name and last name, so that you can sort, search, and index on just those columns. Similarly, the address actually consists of five separate components, address, city, state, postal code, and country/region, and it also makes sense to store them in separate columns. If you want to perform a search, filter or sort operation by state, for example, you need the state information stored in a separate column.

You should also consider whether the database will hold information that is of domestic origin only, or international, as well. For instance, if you plan to store international addresses, it is better to have a Region column instead of State, because such a column can accommodate both domestic states and the regions of other countries/regions. Similarly, Postal Code makes more sense than Zip Code if you are going to store international addresses.

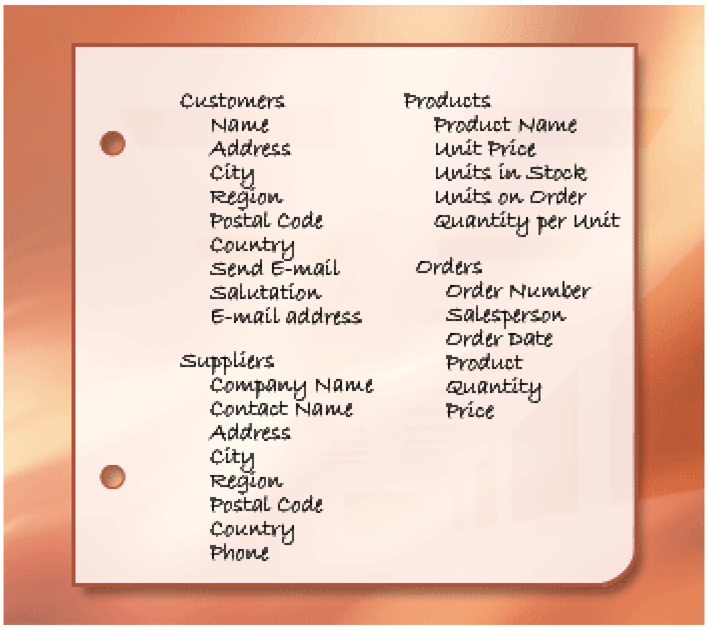
The following list shows a few tips for determining your columns.

- **Don't include calculated data**

In most cases, you should not store the result of calculations in tables. Instead, you can have Access perform the calculations when you want to see the result. For example, suppose there is a Products On Order report that displays the subtotal of units on order for each category of product in the database. However, there is no Units On Order subtotal column in any table. Instead, the Products table includes a Units On Order column that stores the units on order for each product. Using that data, Access calculates the subtotal each time you print the report. The subtotal itself should not be stored in a table.

- **Store information in its smallest logical parts**

You may be tempted to have a single field for full names, or for product names along with product descriptions. If you combine more than one kind of information in a field, it is difficult to retrieve individual facts later. Try to break down information into logical parts; for example, create separate fields for first and last name, or for product name, category, and description.



Customers
- Name
- Address
- City
- Region
- Postal Code
- Country
- Send E-mail
- Salutation
- E-mail address

Suppliers
- Company Name
- Contact Name
- Address
- City
- Region
- Postal Code
- Country
- Phone

Products
- Product Name
- Unit Price
- Units in Stock
- Units on Order
- Quantity per Unit

Orders
- Order Number
- Salesperson
- Order Date
- Product
- Quantity
- Price

Once you have refined the data columns in each table, you are ready to choose each table's primary key.

Specifying primary keys

Each table should include a column or set of columns that uniquely identifies each row stored in the table. This is often a unique identification number, such as an employee ID number or a serial number. In database terminology, this information is called the **primary key** of the table. Access uses primary key fields to quickly associate data from multiple tables and bring the data together for you.

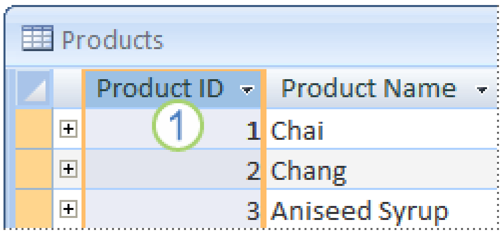
If you already have a unique identifier for a table, such as a product number that uniquely identifies each product in your catalog, you can use that identifier as the table’s primary key — but only if the values in this column will always be different for each record. You cannot have duplicate values in a primary key. For example, don’t use people’s names as a primary key, because names are not unique. You could easily have two people with the same name in the same table.

A primary key must always have a value. If a column's value can become unassigned or unknown (a missing value) at some point, it can't be used as a component in a primary key.

You should always choose a primary key whose value will not change. In a database that uses more than one table, a table's primary key can be used as a reference in other tables. If the primary key changes, the change must also be applied everywhere the key is referenced. Using a primary key that will not change reduces the chance that the primary key might become out of sync with other tables that reference it.

Often, an arbitrary unique number is used as the primary key. For example, you might assign each order a unique order number. The order number's only purpose is to identify an order. Once assigned, it never changes.

If you don't have in mind a column or set of columns that might make a good primary key, consider using a column that has the AutoNumber data type. When you use the AutoNumber data type, Access automatically assigns a value for you. Such an identifier is factless; it contains no factual information describing the row that it represents. Factless identifiers are ideal for use as a primary key because they do not change. A primary key that contains facts about a row — a telephone number or a customer name, for example — is more likely to change, because the factual information itself might change.

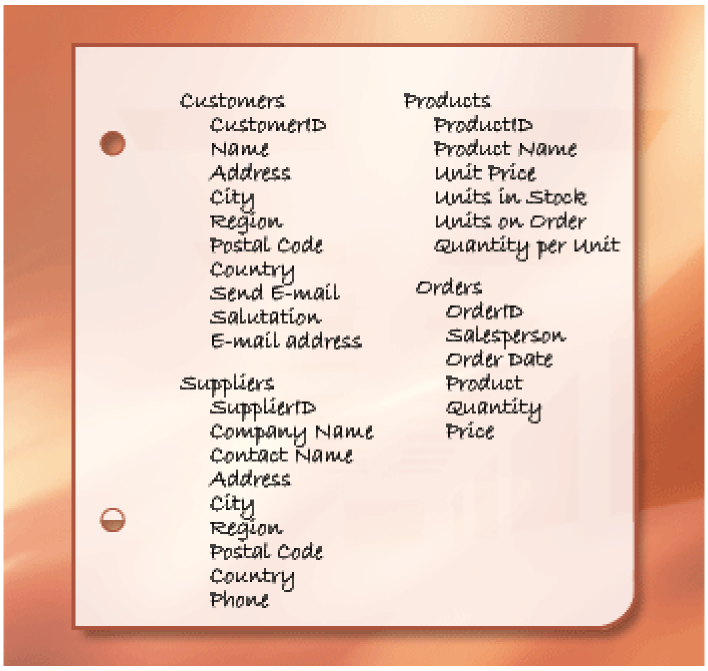


Product ID	Product Name
1	Chai
2	Chang
3	Aniseed Syrup

1 A column set to the AutoNumber data type often makes a good primary key. No two product IDs are the same.

In some cases, you may want to use two or more fields that, together, provide the primary key of a table. For example, an Order Details table that stores line items for orders would use two columns in its primary key: Order ID and Product ID. When a primary key employs more than one column, it is also called a composite key.

For the product sales database, you can create an AutoNumber column for each of the tables to serve as primary key: ProductID for the Products table, OrderID for the Orders table, CustomerID for the Customers table, and SupplierID for the Suppliers table.



Customers
- CustomerID
- Name
- Address
- City
- Region
- Postal Code
- Country
- Send E-mail
- Salutation
- E-mail address

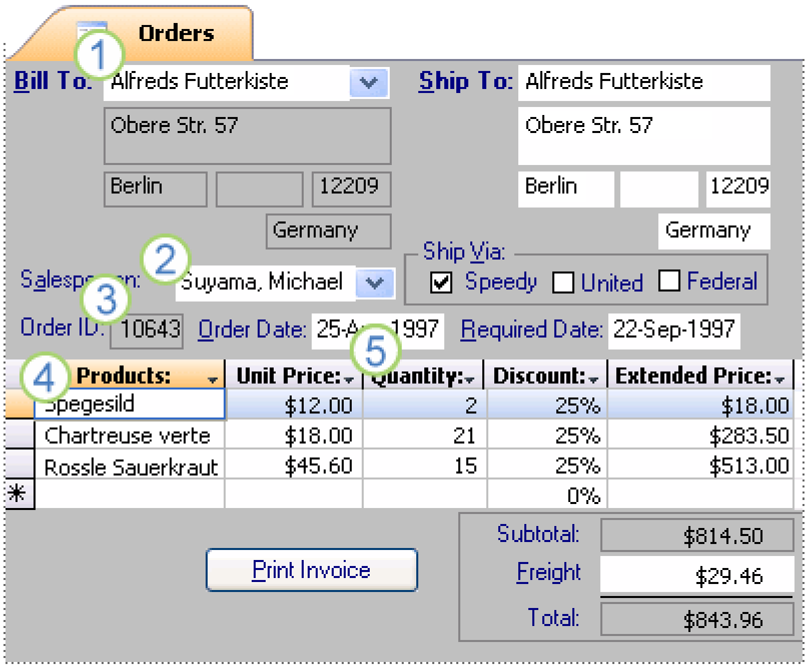
Suppliers
- SupplierID
- Company Name
- Contact Name
- Address
- City
- Region
- Postal Code
- Country
- Phone

Products
- ProductID
- Product Name
- Unit Price
- Units in Stock
- Units on Order
- Quantity per Unit

Orders
- OrderID
- Salesperson
- Order Date
- Product
- Quantity
- Price

Creating the table relationships

Now that you have divided your information into tables, you need a way to bring the information together again in meaningful ways. For example, the following form includes information from several tables.



Orders

Bill To: Alfreds Futterkiste
Obere Str. 57
Berlin | | 12209
Germany

Ship To: Alfreds Futterkiste
Obere Str. 57
Berlin | | 12209
Germany

Salesp on: Suyama, Michael

Ship Via: [✓ Speedy] United Federal

Order ID: 10643 Order Date: 25-A 1997 Required Date: 22-Sep-1997

Products:	Unit Price:	Quantity:	Discount:	Extended Price:
pegesild	$12.00	2	25%	$18.00
Chartreuse verte	$18.00	21	25%	$283.50
Rossle Sauerkraut	$45.60	15	25%	$513.00
			0%	

Subtotal: $814.50
Freight $29.46
Total: $843.96

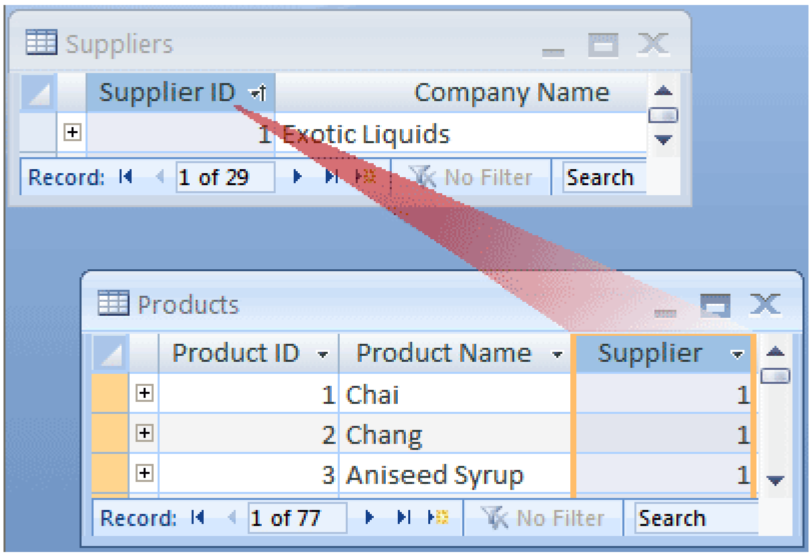
1 Information in this form comes from the Customers table...
2 ...the Employees table...
3 ...the Orders table...
4 ...the Products table...

5 ...and the Order Details table.

Access is a relational database management system. In a relational database, you divide your information into separate, subject-based tables. You then use table relationships to bring the information together as needed.

Creating a one-to-many relationship

Consider this example: the Suppliers and Products tables in the product orders database. A supplier can supply any number of products. It follows that for any supplier represented in the Suppliers table, there can be many products represented in the Products table. The relationship between the Suppliers table and the Products table is, therefore, a one-to-many relationship.

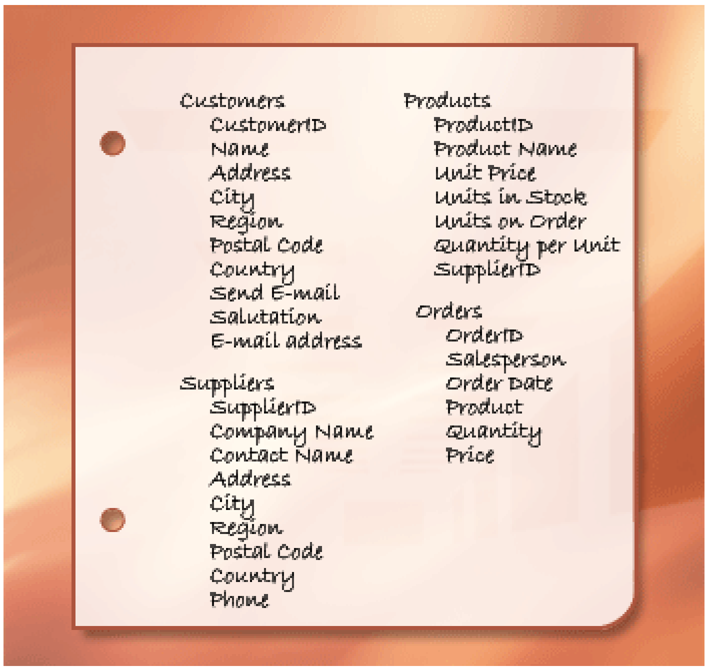


Supplier ID	Company Name
1	Exotic Liquids

Product ID	Product Name	Supplier
1	Chai	1
2	Chang	1
3	Aniseed Syrup	1

To represent a one-to-many relationship in your database design, take the primary key on the "one" side of the relationship and add it as an additional column or columns to the table on the "many" side of the relationship. In this case, for example, you add the Supplier ID column from the Suppliers table to the Products table. Access can then use the supplier ID number in the Products table to locate the correct supplier for each product.

The Supplier ID column in the Products table is called a foreign key. A foreign key is another table’s primary key. The Supplier ID column in the Products table is a foreign key because it is also the primary key in the Suppliers table.



Customers
- CustomerID
- Name
- Address
- City
- Region
- Postal Code
- Country
- Send E-mail
- Salutation
- E-mail address

Suppliers
- SupplierID
- Company Name
- Contact Name
- Address
- City
- Region
- Postal Code
- Country
- Phone

Products
- ProductID
- Product Name
- Unit Price
- Units in Stock
- Units on Order
- Quantity per Unit
- SupplierID

Orders
- OrderID
- Salesperson
- Order Date
- Product
- Quantity
- Price

You provide the basis for joining related tables by establishing pairings of primary keys and foreign keys. If you are not sure which tables should share a common column, identifying a one-to-many relationship ensures that the two tables involved will, indeed, require a shared column.

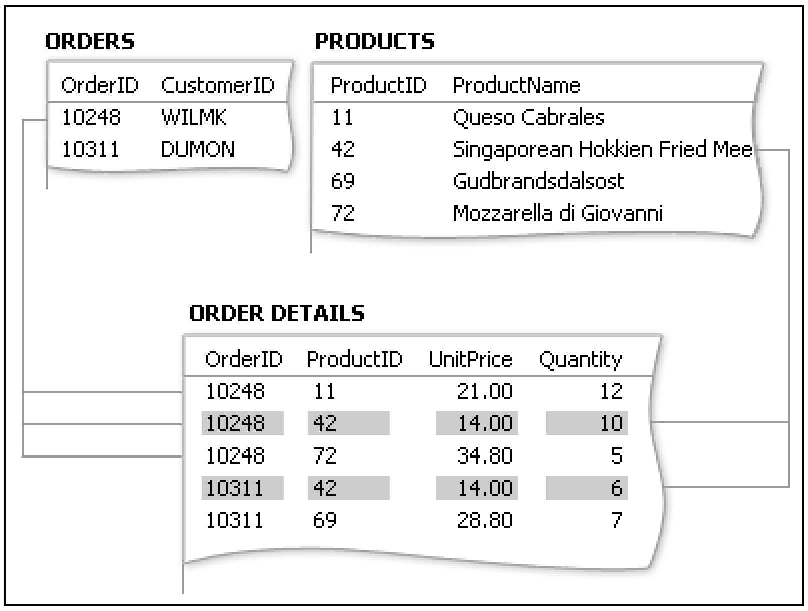
Creating a many-to-many relationship

Consider the relationship between the Products table and Orders table.

A single order can include more than one product. On the other hand, a single product can appear on many orders. Therefore, for each record in the Orders table, there can be many records in the Products table. And for each record in the Products table, there can be many records in the Orders table. This type of relationship is called a many-to-many relationship because for any product, there can be many orders; and for any order, there can be many products. Note that to detect many-to-many relationships between your tables, it is important that you consider both sides of the relationship.

The subjects of the two tables — orders and products — have a many-to-many relationship. This presents a problem. To understand the problem, imagine what would happen if you tried to create the relationship between the two tables by adding the Product ID field to the Orders table. To have more than one product per order, you need more than one record in the Orders table per order. You would be repeating order information for each row that relates to a single order — resulting in an inefficient design that could lead to inaccurate data. You run into the same problem if you put the Order ID field in the Products table — you would have more than one record in the Products table for each product. How do you solve this problem?

The answer is to create a third table, often called a junction table, that breaks down the many-to-many relationship into two one-to-many relationships. You insert the primary key from each of the two tables into the third table. As a result, the third table records each occurrence or instance of the relationship.


ORDERS

OrderID	CustomerID
10248	WILMK
10311	DUMON

PRODUCTS

ProductID	ProductName
11	Queso Cabrales
42	Singaporean Hokkien Fried Mee
69	Gudbrandsdalsost
72	Mozzarella di Giovanni

ORDER DETAILS

OrderID	ProductID	UnitPrice	Quantity
10248	11	21.00	12
10248	42	14.00	10
10248	72	34.80	5
10311	42	14.00	6
10311	69	28.80	7

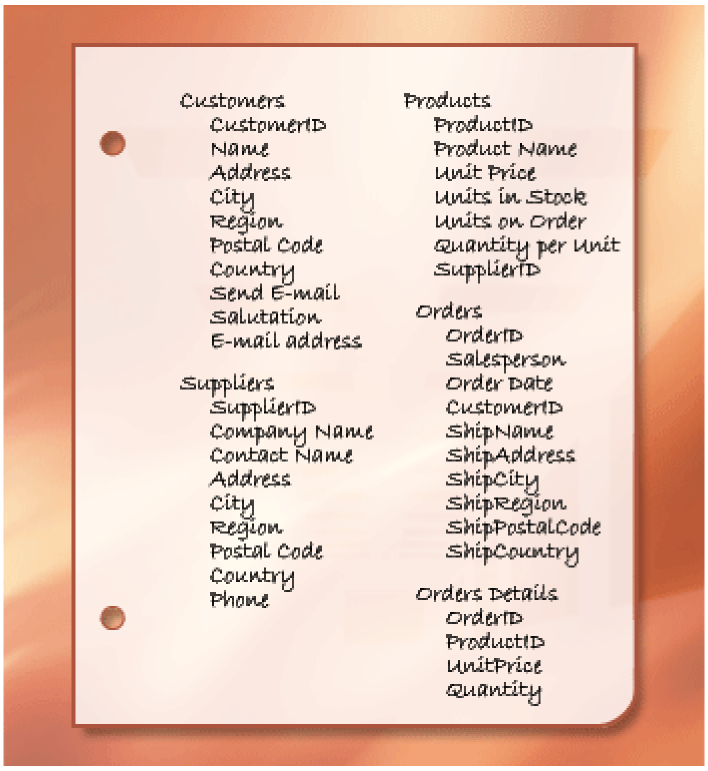
Each record in the Order Details table represents one line item on an order. The Order Details table's primary key consists of two fields — the foreign keys from the Orders and the Products tables. Using the Order ID field alone doesn't work as the primary key for this table, because one order can have many line items. The Order ID is repeated for each line item on an order, so the field doesn't contain unique values. Using the Product ID field alone doesn't work either, because one product can appear on many different orders. But together, the two fields always produce a unique value for each record.

In the product sales database, the Orders table and the Products table are not related to each other directly. Instead, they are related indirectly through the Order Details table. The many-to-many relationship between orders and products is represented in the database by using two one-to-many relationships:

- The Orders table and Order Details table have a one-to-many relationship. Each order can have more than one line item, but each line item is connected to only one order.
- The Products table and Order Details table have a one-to-many relationship. Each product can have many line items associated with it, but each line item refers to only one product.

From the Order Details table, you can determine all of the products on a particular order. You can also determine all of the orders for a particular product.

After incorporating the Order Details table, the list of tables and fields might look something like this:



Customers
- CustomerID
- Name
- Address
- City
- Region
- Postal Code
- Country
- Send E-mail
- Salutation
- E-mail address

Suppliers
- SupplierID
- Company Name
- Contact Name
- Address
- City
- Region
- Postal Code
- Country
- Phone

Products
- ProductID
- Product Name
- Unit Price
- Units in Stock
- Units on Order
- Quantity per Unit
- SupplierID

Orders
- OrderID
- Salesperson
- Order Date
- CustomerID
- ShipName
- ShipAddress
- ShipCity
- ShipRegion
- ShipPostalCode
- ShipCountry

Orders Details
- OrderID
- ProductID
- UnitPrice
- Quantity

Creating a one-to-one relationship

Another type of relationship is the one-to-one relationship. For instance, suppose you need to record some special supplementary product information that you will need rarely or that only applies to a few products. Because you don't need the information often, and because storing the information in the Products table would result in empty space for every product to which it doesn’t apply, you place it in a separate table. Like the Products table, you use the ProductID as the primary key. The relationship between this supplemental table and the Product table is a one-to-one relationship. For each record in the Product table, there exists a single matching record in the supplemental table. When you do identify such a relationship, both tables must share a common field.

When you detect the need for a one-to-one relationship in your database, consider whether you can put the information from the two tables together in one table. If you don’t want to do that for some reason, perhaps because it would result in a lot of empty space, the following list shows how you would represent the relationship in your design:

- If the two tables have the same subject, you can probably set up the relationship by using the same primary key in both tables.
- If the two tables have different subjects with different primary keys, choose one of the tables (either one) and insert its primary key in the other table as a foreign key.

Determining the relationships between tables helps you ensure that you have the right tables and columns. When a one-to-one or one-to-many relationship exists, the tables involved need to share a common column or columns. When a many-to-many relationship exists, a third table is needed to represent the relationship.

Refining the design

Once you have the tables, fields, and relationships you need, you should create and populate your tables with sample data and try working with the information: creating queries, adding new records, and so on. Doing this helps highlight potential problems — for example, you might need to add a column that you forgot to insert during your design phase, or you may have a table that you should split into two tables to remove duplication.

See if you can use the database to get the answers you want. Create rough drafts of your forms and reports and see if they show the data you expect. Look for unnecessary duplication of data and, when you find any, alter your design to eliminate it.

As you try out your initial database, you will probably discover room for improvement. Here are a few things to check for:

- Did you forget any columns? If so, does the information belong in the existing tables? If it is information about something else, you may need to create another table. Create a column for every information item you need to track. If the information can't be calculated from other columns, it is likely that you will need a new column for it.
- Are any columns unnecessary because they can be calculated from existing fields? If an information item can be calculated from other existing columns — a discounted price calculated from the retail price, for example — it is usually better to do just that, and avoid creating new column.
- Are you repeatedly entering duplicate information in one of your tables? If so, you probably need to divide the table into two tables that have a one-to-many relationship.
- Do you have tables with many fields, a limited number of records, and many empty fields in individual records? If so, think about redesigning the table so it has fewer fields and more records.
- Has each information item been broken into its smallest useful parts? If you need to report, sort, search, or calculate on an item of information, put that item in its own column.
- Does each column contain a fact about the table's subject? If a column does not contain information about the table's subject, it belongs in a different table.
- Are all relationships between tables represented, either by common fields or by a third table? One-to-one and one-to- many relationships require common columns. Many-to-many relationships require a third table.

Refining the Products table

Suppose that each product in the product sales database falls under a general category, such as beverages, condiments, or seafood. The Products table could include a field that shows the category of each product.

Suppose that after examining and refining the design of the database, you decide to store a description of the category along with its name. If you add a Category Description field to the Products table, you have to repeat each category description for each product that falls under the category — this is not a good solution.

A better solution is to make Categories a new subject for the database to track, with its own table and its own primary key. You can then add the primary key from the Categories table to the Products table as a foreign key.

The Categories and Products tables have a one-to-many relationship: a category can include more than one product, but a product can belong to only one category.

When you review your table structures, be on the lookout for repeating groups. For example, consider a table containing the following columns:

- Product ID
- Name
- Product ID1
- Name1
- Product ID2
- Name2
- Product ID3
- Name3

Here, each product is a repeating group of columns that differs from the others only by adding a number to the end of the column name. When you see columns numbered this way, you should revisit your design.

Such a design has several flaws. For starters, it forces you to place an upper limit on the number of products. As soon as you exceed that limit, you must add a new group of columns to the table structure, which is a major administrative task.

Another problem is that those suppliers that have fewer than the maximum number of products will waste some space, since the additional columns will be blank. The most serious flaw with such a design is that it makes many tasks difficult to perform, such as sorting or indexing the table by product ID or name.

Whenever you see repeating groups review the design closely with an eye on splitting the table in two. In the above example it is better to use two tables, one for suppliers and one for products, linked by supplier ID.

Applying the normalization rules

You can apply the data normalization rules (sometimes just called normalization rules) as the next step in your design. You use these rules to see if your tables are structured correctly. The process of applying the rules to your database design is called normalizing the database, or just normalization.

Normalization is most useful after you have represented all of the information items and have arrived at a preliminary design. The idea is to help you ensure that you have divided your information items into the appropriate tables. What normalization cannot do is ensure that you have all the correct data items to begin with.

You apply the rules in succession, at each step ensuring that your design arrives at one of what is known as the "normal forms." Five normal forms are widely accepted — the first normal form through the fifth normal form. This article expands on the first three, because they are all that is required for the majority of database designs.

First normal form

First normal form states that at every row and column intersection in the table there, exists a single value, and never a list of values. For example, you cannot have a field named Price in which you place more than one Price. If you think of each intersection of rows and columns as a cell, each cell can hold only one value.

Second normal form

Second normal form requires that each non-key column be fully dependent on the entire primary key, not on just part of the key. This rule applies when you have a primary key that consists of more than one column. For example, suppose you have a table containing the following columns, where Order ID and Product ID form the primary key:

- Order ID (primary key)
- Product ID (primary key)
- Product Name

This design violates second normal form, because Product Name is dependent on Product ID, but not on Order ID, so it is not dependent on the entire primary key. You must remove Product Name from the table. It belongs in a different table (Products).

Third normal form

Third normal form requires that not only every non-key column be dependent on the entire primary key, but that non-key columns be independent of each other.

Another way of saying this is that each non-key column must be dependent on the primary key and nothing but the primary key. For example, suppose you have a table containing the following columns:

- ProductID (primary key)
- Name
- SRP
- Discount

Assume that Discount depends on the suggested retail price (SRP). This table violates third normal form because a non-key column, Discount, depends on another non-key column, SRP. Column independence means that you should be able to change any non-key column without affecting any other column. If you change a value in the SRP field, the Discount would change accordingly, thus violating that rule. In this case Discount should be moved to another table that is keyed on SRP.