# Keys and Relationships

## Working with keys and relationships

- As you've seen, normalization is the process of identifying and separating out potentially redundant data into separate tables in your database.

- Once you've identified the data that you want to normalize, you'll need to isolate that data within its own table. Then, you’ll need to establish a relationship between the new table and the parent table using a series of **keys and relationships**. This is known as **relationship management**.

# Keys and Relationships

## Working with keys and relationships

Consider the following two tables. In the last lecture you learned that it would make more sense to break out the employee data into its own table to avoid duplication.

| Tickets |
|---|
| ticketid |
| issue |
| customername |
| customeremail |
| submitteddate |

| Employees |
|---|
| employeeid |
| name |
| username |
| password |
| email |

For every ticket that is generated in our database, we'll need an employee assigned to that ticket. Although there will only ever be one unique ticket, there might be hundreds of tickets assigned to a specific employee. To avoid duplication and improve performance, we create an employee table and make reference to that employee via the tickets table.

# Keys and Relationships

## Working with keys and relationships

In the previous example, the employees and tickets tables are related. Every employee within the company might support dozens, perhaps hundreds of customers and their associated tickets within the tickets table. This would be an example of a **one-to-many relationship**. While the one-to-many relationship is the most common, it's not the only type. There are several types of relationships, including:

- One to One Relationships
- One to Many and Many to One Relationships
- Many to Many Relationships
- Self Referencing Relationships

Let's look at each…

# Keys and Relationships

## One-to-one relationships

To help you understand one-to-one relationships, consider the following Customers table:

| id | name | address |
|---|---|---|
| 101 | Sally Smith | 555 Main St. Houston TX 77001 |
| 102 | Mark Jones | 321 E. Main Gotham NY 10286 |
| 103 | Frank Davidson | 21 Pine St. Beverly Hills CA 90210 |

| id | name | aid |
|---|---|---|
| 101 | Sally Smith | 1 |
| 102 | Mark Jones | 2 |
| 103 | Frank Davidson | 3 |

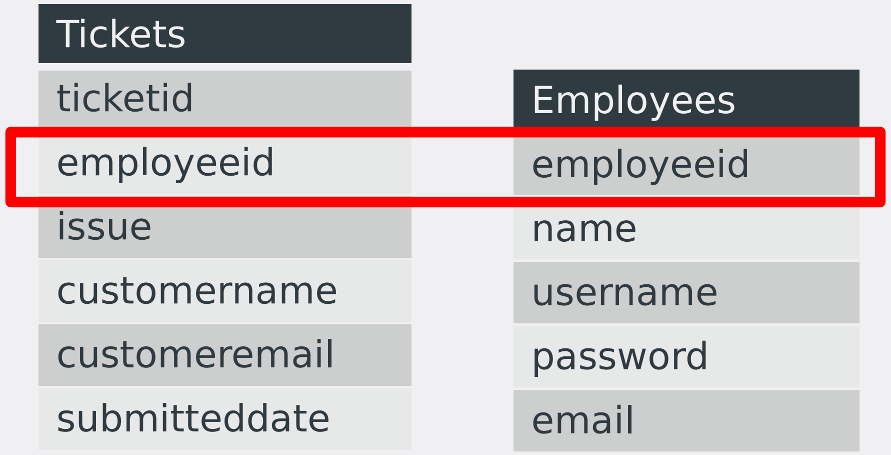
| aid | street | city | state | zip |
|---|---|---|---|---|
| 1 | 555 Main St. | Houston | TX | 77001 |
| 2 | 321 E. Main | Gotham | NY | 10286 |
| 3 | 21 Pine St. | Beverly Hills | CA | 90210 |

Now we have a relationship between the customers table and the addresses table. If each address can belong

# Keys and Relationships

## One-to-many relationships

This is the most commonly used type of relationship. Again, consider our tickets / employees relationship. For every employee that we have within our organization, she'll undoubtedly have numerous tickets to take care of. With that said, our previously created tables would begin to look like this:



| Tickets |
|---|
| ticketid |
| employeeid |
| issue |
| customername |
| customeremail |
| submitteddate |

| Employees |
|---|
| employeeid |
| name |
| username |
| password |
| email |

In this scenario, the relationship is established between the employees and tickets tables via the employeeid field. More on this in a bit...

# Keys and Relationships

## Many-to-many relationships

In some cases, you may need multiple instances on both sides of the relationship. For example, consider an orders table and a products table. A report might need to be generated that outlined how many orders were generated for a particular product per day. At the same time, an order can contain multiple products. The structure would begin to look like this:

| orderid | customerid | date |
|---|---|---|
| 111 | 101 | 04/28/2018 |
| 112 | 102 | 04/29/2018 |

| productid | name | description | cost |
|---|---|---|---|
| 0001 | Widget | This is a widget | $30.22 |
| 0002 | Gadget | This is a gadget | $156.18 |

| orderid | productid |
|---|---|
| 111 | 0001 |
| 111 | 0001 |
| 111 | 0002 |
| 112 | 0001 |
| 112 | 0002 |

The table on the right (let's call it the ProductOrders table) has only one purpose, and that is to create a "Many

# Keys and Relationships

## Self referencing relationships

Self referencing relationships are used when a table needs to have a relationship with itself. For example, let's say a fictitious store has a referral program. Customers can refer other customers to the ecommerce website. The table may look like this:

| customerid | name | referrer_customerid |
|---|---|---|
| 0001 | Sally Smith | |
| 0002 | John Doe | 0001 |
| 0003 | Mark Martinson | 0001 |
| 0004 | Alex Payne | 0001 |
| 0005 | Desiree Watkins | |

Customers 0002, 0003, and 0004 were referred by the customer 0001. This actually can also be similar to a "one to many" relationship since one customer can refer multiple customers.

# Keys and Relationships

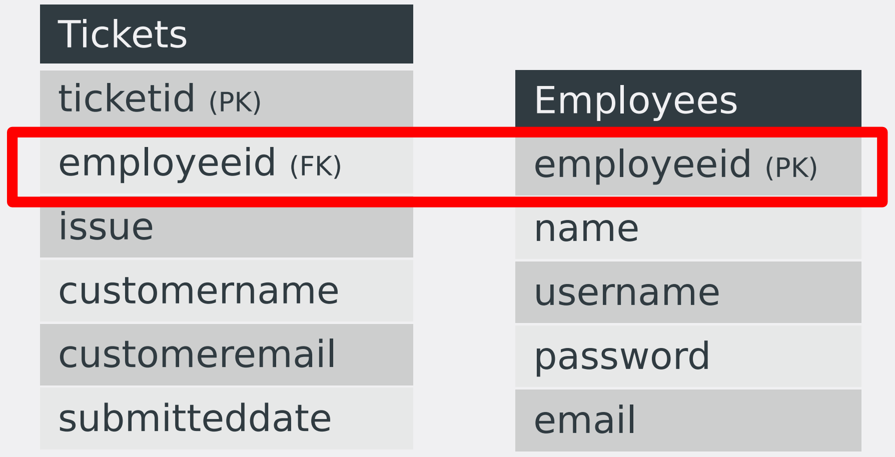
## Keys

- Creating relationships in a database begins with keys: **primary keys** and **foreign keys**.
- In a one-to-many relationship for instance, a primary key identifies a record (usually numeric) in a table as being the parent (primary) in the one-to-many relationship.
- In our tickets > employees relationship, the employeeid in the employees table would be the primary key while the same employeeid column in the tickets table would be the foreign key.
- Essentially this translates to: for every one employee in our employees table, they will have many tickets assigned to them in the tickets table.

# Keys and Relationships

## Keys

Our one-to-many relationship would end up looking like this:



| Tickets |
|---|
| ticketid (PK) |
| employeeid (FK) |
| issue |
| customername |
| customeremail |
| submitteddate |

| Employees |
|---|
| employeeid (PK) |
| name |
| username |
| password |
| email |

# Keys and Relationships

## Keys

With the relationship now established between the employees / tickets tables, we can now set up the relationships for the other tables in our database. The implementation would begin to look like this:

| **Primary Key Table** | **Foreign Key** | **Foreign Key Table** |
|---|---|---|
| employees | employeeid | tickets |
| status | statusid | tickets |
| solutions | solutionid | tickets |
| roles | roleid | employees |