Section slides: http://webdev.slides.com/coltsteele/mysql-99-104

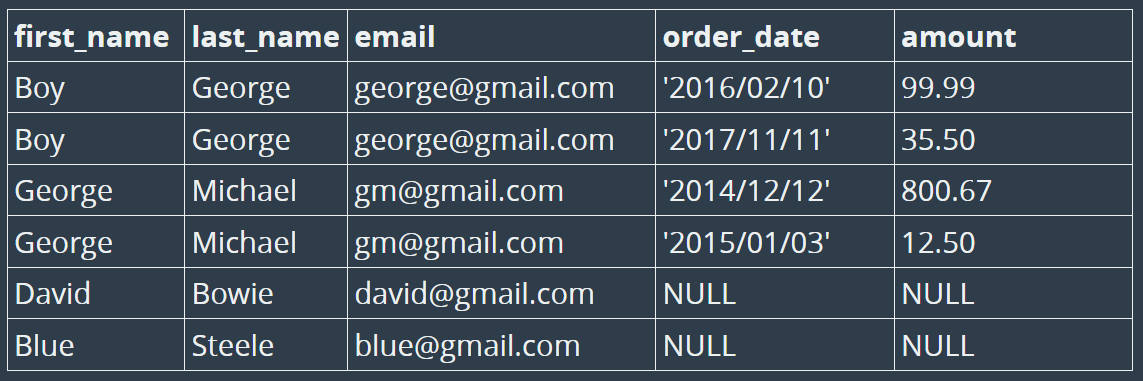
* In this section we will see how to relate tables to one another
* Prior to now, we were working with relatively simple data in only a single table, and those tables had only a few columns
* But in the real world, we can very complex data that is housed across multiple tables, and this inter-tabular data tends to be interrelated and dependent on each other
* Storing important information for a typical database may require half a dozen or more tables

# Types of Data Relationships

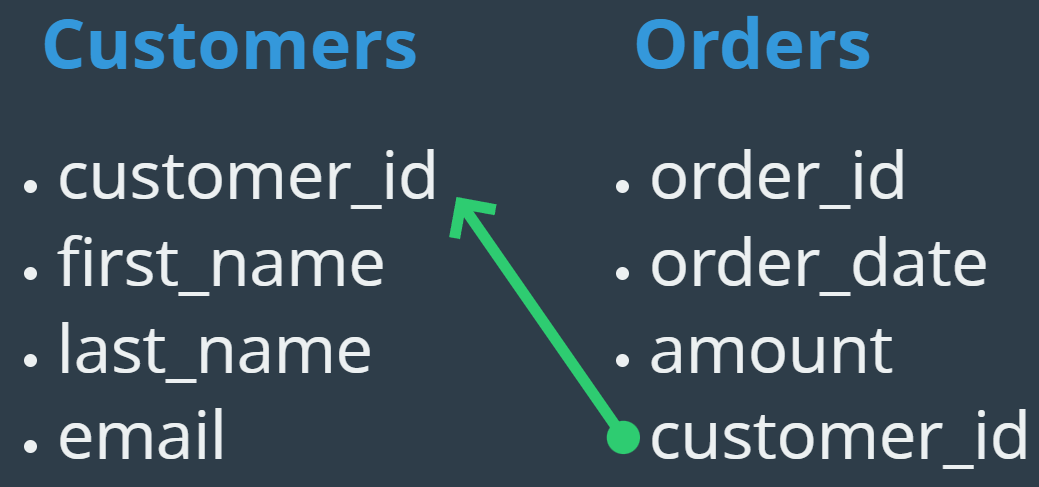
* In SQL, there are three broad categories of relationships
  + One to One: when each entry in one table is related to one and only one entry in another table
    - Ex: Customer ID Table to Customer Details table
  + One to Many: when one entry in one table is related to zero, one, or multiple entries in another table
    - Ex: A table of reviews for books to a table of books
    - The table of reviews can have multiple reviews for one book, but all of those reviews belong only to that one book
  + Many to Many: when many entries in one table are related to zero, one, or multiple entries in another table
    - Ex: In a database of books and authors, a given book can have multiple authors (books table), and each of those authors can be authoring multiple books independently (authors table)
  + More examples of relationships: <https://condor.depaul.edu/gandrus/240IT/accesspages/relationships.htm#:~:text=There%20are%20three%20types%20of,to%20the%20data%20and%20tables>.

# The Basics of One To Many

* 1:Many is perhaps the most common type of relationship in relational databases
* Consider an example situation where we have two tables: one of customers, and one of orders
  + Customers can have more than one order, but a given only has one customer associated with it
  + We want to store
    - A customer’s first and last names (VARCHAR)
    - A customer’s email (VARCHAR)
    - The date of the purchase (DATE)
    - The price of the order (DECIMAL)
* What would be a way to approach storing this data? One option is a single table where we capture every order. Notice how some customers have more than one order



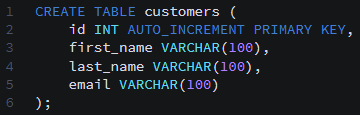
* + However, this is NOT a good idea.
    - There is duplication of some data, line names and emails
    - Some customers have not placed any orders, so those values are NULL. There is no reason to have data on orders for customers who have not placed any orders. Instead, it would be better to have customers tracked in a separate table
* The simplest approach to this issue is to create *two separate tables* for **Customers** and **Orders**, and they will be related to each other through **customer\_id**, which will be a column in BOTH tables
  + As a result of this reorganization, each customer in the *customers* table is unique and has a dedicated **customer\_id**
  + The *orders* table also has a **customer\_id** column – note that because a customer can place more than one order, a given customer\_id can appear more than once in this table. Similarly, any given order is associated with one and only one customer
  + Therefore, the relationship between these two tables is One (customers) to Many (Orders)
  + We also see that the customers who have not placed any orders are no long associated with NULL values as they were in the single table
  + Both tables have a PRIMARY KEY, which is a column that contains NO duplicate values and uniquely identifies that row of data
    - For the *customers* table, the PRIMARY KEY is **customer\_id**
    - For the *orders* table, the PRIMARY KEY is **order\_id**
  + Here we see our first **FOREIGN KEY**. A foreign key is a key that references the PRIMARY KEY of another table.
    - For our *orders* table, the FOREIGN KEY is **customer\_id**, as it refers to the customer\_id value in the *customers* table
    - *customers* does NOT have a FOREIGN KEY, as it does not have a column that references a PRIMARY KEY of the *orders* table
  + Foreign keys are important for database integrity. For example, we would not be able to add data to the *orders* table with a customer\_id that does not exist in the *customers* table



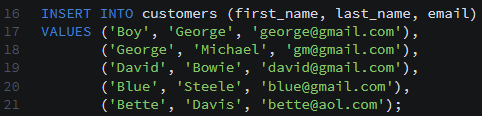


# Working with Foreign Keys

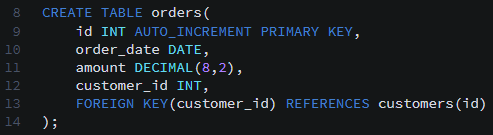
* For this section, we’ll be creating our *customers* and *orders* tables to demonstrate working with foreign keys. We’ll do this within a new database called **customers\_and\_orders**
* For *customers*, we’ll have the columns customer\_id, first\_name\_last\_name, and email. customer\_id will be an INT with auto-increment as well as our primary key, while the remainder will be VARCHAR
  + We will be referencing customer\_id as a foreign key from another table, and thus it must be unique

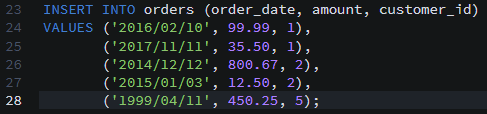


* + Inserting our data



* For *orders*, we’ll have id, order\_date, amount, and customer\_id. id will be an INTO and primary key, order\_date will be a DATE, and amount will be DECIMAL, and customer\_id will be INT FOREIGN KEY.
  + FOREIGN KEY is a new concept for us. The syntax involves using the REFERENCES keyword in our table creation, and ensure that the reference points to the customer\_id column of the *customers* table (see syntax below)
  + Why do we need to use FOREIGN KEY? The reason is that customer\_id is how we will be linking *customers* and *orders* together, and thus for the purposes of database integrity, any customer\_id that appears in *orders* must also exist in *customers*.
    - Important note: The variable names of the FOREIGN KEY and the PRIMARY KEY that it references do not need to be the same. In our example, customer\_id is simply “id” in the *customers* table but is “customer\_id” in the *orders* table
    - This is conventional when using a foreign key





* To test our database integrity, let’s try to insert an order for a customer that does not exist
  + The insertion will fail because we have no customer whose ID is 98





* We now have these tables associated with each other. In the next table we’ll explore JOINS, where we can do things like get the name of the customer based on orders they placed.