PostgreSQL and pgAdmin. He'll use Postgres to create a database, and pgAdmin to work with the data he'll be importing. These tools are packaged together in a single download.

**PostgreSQL**

PostgreSQL, typically referred to as just "Postgres," is a **relational database system**. This type of database consists of tables and their predefined relationships.

## pgAdmin

pgAdmin is the window into our database: it's where queries are written and executed and where results are viewed. While Postgres holds the files, pgAdmin provides the access. All SQL actions take place within these two programs, so let's install them.

## Entity Relationship Diagrams (ERDs)

An **entity relationship diagram (ERD)** is a type of flowchart that highlights different tables and their relationships to each other. The ERD does not include any actual data, but it does capture the following pertinent information from each CSV file:

* Primary keys
* Foreign keys
* Data types for each column

Database components include tables, known as **entities,** with data, known as **attributes.** Data types include Booleans, integers, and varying characters (i.e., within a string).

There are three types of ERDs: **conceptual, logical,** and **physical.** Each one builds upon the other—you need the conceptual ERD to build a logical ERD to build a physical ERD.

we're ready to use our new tool: Quick DBD. <https://app.quickdatabasediagrams.com/#/>

There are several ways to refer to the map we're about to create. It's also called a flowchart, an entity relationship diagram, and a schema. We'll be using all of these terms in this module, though "ERD" is the most specific.  
  
There are three forms of ERDs: conceptual, logical, and physical.

## Conceptual Diagrams

A **conceptual diagram** is an ERD in its simplest form. To create one, we only need two things: a table name and column headers.

It's simple because we're creating just the concept of the diagram.

The next step in building a diagram is to assign data types, which transitions our conceptual ERD to a logical ERD.

## Logical Diagrams

**Logical diagrams** contain all of the same information that a conceptual diagram does, but the table is updated to include data types and primary keys.

Next, add foreign keys and physically map the relationships between them.

## Physical Diagrams

**Physical diagrams** portray the physical relationship, or how the data is connected, between each table.

Now that **You've** created an entire map of a database! Let’s create a database in Postgres. Postgres is where our data will actually live once we import it.

What do you think would happen if one of the rows didn’t have a unique identifier?

* 

Nothing, all of the queries would still work correctly.

Nothing, all of the queries would still work correctly.

* 

None of the queries would work.

None of the queries would work.

* 

All of the queries would work, but the data might not be included. That’s okay though, because what’s one line of data?

All of the queries would work, but the data might not be included. That’s okay though, because what’s one line of data?

* 

Not all of the data would be present in every query, which would skew analysis results and provide incomplete lists.

Not all of the data would be present in every query, which would skew analysis results and provide incomplete lists.

Correct. Nice work! Not requiring data in such an important field leaves a vulnerability in the data.

Notice that the table names and column names are written in different cases. Table names are typically written in lowercase, while the rest of the statement is written in uppercase. This helps differentiate between table and column names as well as commands and parameters.