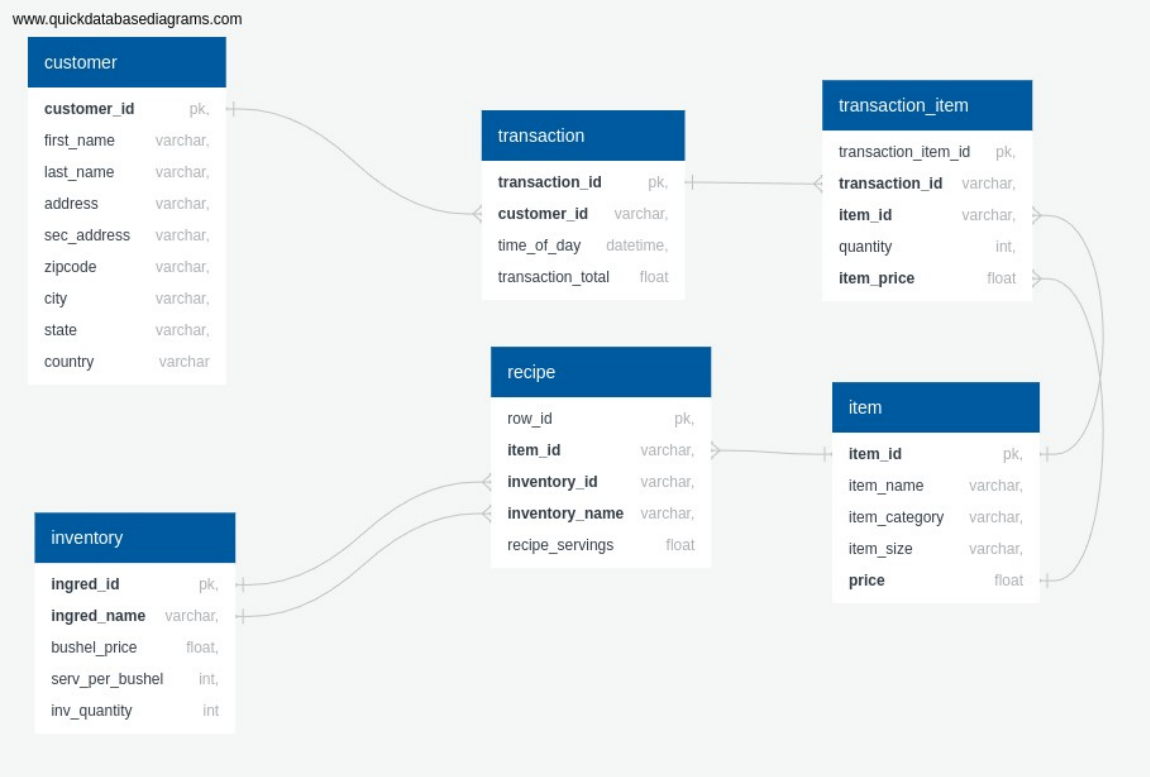
Kickoff Design Database:

This document explains general principles behind designing a postgreSQL database, and intentionally does not explain why the database was designed the way that it was to allow room for change and growth within the database.



- First, it's important to understand the purpose and intent of the database before beginning to design it. Understanding purpose and intent means understanding what questions need to be answered and how your database and tables will help you to answer those questions.

- It's best to use a .sql script to create the database instead of working line by line in the psql shell to create your database for a number of reasons. With a .sql script, you can: return to your database in the future and make improvements to your database design, understand the schema, or completely restart if necessary.

- Utilizing a tool like quickdatabasediagrams.com , or QuickDBD (photo at the top) allows you to visualize how all of your tables relate and connect to one another. While Quick DBD will even auto-generate the SQL code you need in a .sql file, I chose not to utilize this feature for this project to gain skill and familiarity with how SQL Databases are designed.

- Looking closely at the diagram of the database schema, you can see the lines that connect one table to another. One end of each line ends with a 3 prong shape and the opposite end has a single endpoint. Which table the 3-prong or single endpoint connect to has significant meaning. The single endpoint of the line connects to the parent table/column while the 3-prong connects to a child.

For example, take the line that connects customer_id on the 'customer' table to customer_id on the 'transaction' table. The customer_id field in the 'transaction' table won't be a unique value in order to allow the database to record individual transactions by repeat customers. The 'customer' table is the parent, while the 'transaction' table is the child. This also highlights how in many instances, the lines connect are governed by a rule I call "The many to the one". The parent column being referenced will typically have unique values, while the child being reference won't have unique vales. The customer_id example, highlights this

rule as well as the item(item_id) → transaction_item(item_id) connection. In the item(item_id) → transaction_item(item_id) connection, item(item_id) will be a unique value in that table that non-unique values in transaction_item(item_id) reference.